Sintaxis y procesamiento de cifrado XML Versión 1.1

Recomendación del W3C 11 de abril de 2013

Esta versión:
http://www.w3.org/TR/2013/REC-xmlenc-core1-20130411/
Última versión publicada:
http://www.w3.org/TR/xmlenc-core1/
Último borrador del editor:
http://www.w3.org/2008/xmlsec/Drafts/xmlenc-core-11/
Versión previa:
http://www.w3.org/TR/2013/PR-xmlenc-core1-20130124/
Editores:
Donald Eastlake , d3e3e3@gmail.com
Joseph Reagle , reagle@mit.edu
Federico Hirsch , frederick.hirsch@nokia.com (1.1)
Thomas Roessler , tlr@w3.org (1.1)
Autores:
Takeshi Imamura , IMAMU@jp.ibm.com
Blair Dillaway , blaird@microsoft.com
Ed Simon , edsimon@xmlsec.com
Kelvin Yiu , kelviny@microsoft.com (1.1)
Magnus Nyström , mnystrom@microsoft.com (1.1)

Consulte las erratas de este documento, que pueden incluir algunas correcciones normativas.

La versión en inglés de esta especificación es la única versión normativa. También pueden estar disponibles traducciones no normativas .

Copyright © 2013 W3C ® (MIT , ERCIM , Keio , Beihang), Todos los derechos reservados. Se aplican las reglas de responsabilidad , marcas comerciales y uso de documentos del W3C .

Abstracto

Este documento especifica un proceso para cifrar datos y representar el resultado en XML. Los datos pueden estar en una variedad de formatos, incluidos flujos de octetos y otros datos no estructurados, o formatos de datos estructurados como documentos XML, un elemento XML o contenido de un elemento XML. El resultado del cifrado de datos es un elemento de cifrado XML que contiene o hace referencia a los datos cifrados.

Estado de este documento

Esta sección describe el estado de este documento en el momento de su publicación. Otros documentos pueden reemplazar este documento. Puede encontrar una lista de las publicaciones actuales del W3C y la última revisión de este informe técnico en el índice de informes técnicos del W3C en http://www.w3.org/TR/.

Este documento ha sido revisado por miembros del W3C , desarrolladores de software y otros grupos del W3C y partes interesadas, y cuenta con el respaldo del Director como Recomendación del W3C . Es un documento estable y puede usarse como material de referencia o citarse de otro documento. El papel del W3C al elaborar la Recomendación es llamar la atención sobre la especificación y promover su implementación generalizada. Esto mejora la funcionalidad y la interoperabilidad de la Web.

La versión original de esta especificación fue producida por el Grupo de Trabajo de Cifrado XML del W3C ; El Informe de interoperabilidad muestra cuatro implementaciones con al menos dos implementaciones interoperables en cada característica.

Consulte el informe de implementación de la versión 1.1 de esta especificación para obtener detalles adicionales sobre el estado de implementación de las funciones agregadas en esta revisión.

Los cambios que afectan la conformidad con respecto a la Recomendación anterior afectan principalmente al conjunto de algoritmos criptográficos obligatorios para implementar, agregando el Acuerdo de Clave Diffie-Hellman de Curva Elíptica, haciendo obligatorio AES-128 GCM, cambiando RSA v1.5 a opcional, agregando AES192-GCM opcional y agregando variantes opcionales del algoritmo RSA-OEAP. También se han agregado importantes consideraciones de seguridad. Un resumen detallado de los cambios está disponible en [XMLENC-CORE1-CHGS]. Los cambios también se describen en un documento de diferencias que muestra los cambios desde la Recomendación original , así como en un documento de diferencias que muestra los cambios desde el borrador de PR anterior .

Este documento fue publicado por el Grupo de Trabajo de Seguridad XML como recomendación. Si desea hacer comentarios sobre este documento, envíelos a public-xmlsec@w3.org (suscríbase , archivos). Todos los comentarios son bienvenidos.

Este documento fue elaborado por un grupo que opera bajo la Política de Patentes del W3C del 5 de febrero de 2004 . El W3C mantiene una lista pública de cualquier divulgación de patentes realizada en relación con los productos del grupo; esa página también incluye instrucciones para divulgar una patente. Una persona que tenga conocimiento real de una patente que cree que contiene Reivindicaciones Esenciales debe revelar la información de acuerdo con la sección 6 de la Política de Patentes del W3C .

También está disponible información adicional relacionada con el estado de los derechos de propiedad intelectual de XML Encryption 1.1 .

Tabla de contenido

1. Introduction

This document specifies a process for encrypting data and representing the result in XML. The data may be arbitrary data (including an XML document), an XML element, or XML element content. The result of encrypting data is an XML Encryption `EncryptedData` element that contains (via one of its children's content) or identifies (via a URI reference) the cipher data.

When encrypting an XML element or element content the `EncryptedData` element replaces the element or content (respectively) in the encrypted version of the XML document.

When encrypting arbitrary data (including entire XML documents), the `EncryptedData` element may become the root of a new XML document or become a child element in an application-chosen XML document.

1.1 Editorial and Conformance Conventions

This specification uses XML schemas [XMLSCHEMA-1], [XMLSCHEMA-2] to describe the content model. The full normative grammar is defined by the XSD schema and the normative text in this specification. The standalone XSD schema file is authoritative in case there is any disagreement between it and the XSD schema portions.

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this specification are to be interpreted as described in [RFC2119]:

> "They MUST only be used where it is actually required for interoperation or to limit behavior which has potential for causing harm (e.g., limiting retransmissions)"

Consequently, we use these capitalized keywords to unambiguously specify requirements over protocol and application features and behavior that affect the interoperability and security of implementations. These key words are not used (capitalized) to describe XML grammar; schema definitions unambiguously describe such requirements and we wish to reserve the prominence of these terms for the natural language descriptions of protocols and features. For instance, an XML attribute might be described as being "optional". Compliance with the XML-namespace specification [XML-NAMES] is described as "REQUIRED".

1.2 Design Philosophy

The design philosophy and requirements of this specification (including the limitations related to instance validity) are addressed in the original XML Encryption Requirements [XML-ENCRYPTION-REQ] and the XML Security 1.1 Requirements document [XMLSEC11-REQS].

1.3 Versions, Namespaces, URIs, and Identifiers

This specification makes use of XML namespaces, and uses Uniform Resource Identifiers [URI] to identify resources, algorithms, and semantics.

Implementations of this specification MUST use the following XML namespace URIs:

URI	namespace prefix	XML internal entity
`http://www.w3.org/2001/04/xmlenc#`	*default namespace*, `xenc:`	`<!ENTITY xenc "http://www.w3.org/2001/04/xmlenc#">`
`http://www.w3.org/2009/xmlenc11#`	`xenc11:`	`<!ENTITY xenc11 "http://www.w3.org/2009/xmlenc11#">`

The `http://www.w3.org/2001/04/xmlenc#` (`xenc:`) namespace was introduced in version 1.0 of this specification. The present version does not coin any new elements or algorithm identifiers in that namespace; instead, the `http://www.w3.org/2009/xmlenc11#` (`xenc11:`) namespace is used.

No provision is made for an explicit version number in this syntax. If a future version of this specification requires explicit versioning of the document format, a different namespace will be used.

Additionally, this specification uses elements and algorithm identifiers from the XML Signature name spaces [XMLDSIG-CORE1]:

URI	namespace prefix	XML internal entity
`http://www.w3.org/2000/09/xmldsig#`	*default namespace*, `ds:`, `dsig:`	`<!ENTITY dsig "http://www.w3.org/2000/09/xmldsig#">`
`http://www.w3.org/2009/xmldsig11#`	`dsig11:`	`<!ENTITY dsig11 "http://www.w3.org/2009/xmldsig11#">`

1.4 Acknowledgements

The contributions of the following members of the original Working Group to the original XML Encryption specification are gratefully acknowledged in accordance with the [contributor policies](#) and the active [WG roster](#): Joseph Ashwood, Simon Blake-Wilson, Certicom, Frank D. Cavallito, BEA Systems, Eric Cohen, PricewaterhouseCoopers, Blair Dillaway, Microsoft (Author), Blake Dournaee, RSA Security, Donald Eastlake, Motorola (Editor), Barb Fox, Microsoft, Christian Geuer-Pollmann, University of Siegen, Tom Gindin, IBM, Jiandong Guo, Phaos, Phillip Hallam-Baker, Verisign, Amir Herzberg, NewGenPay, Merlin Hughes, Baltimore, Frederick Hirsch, Maryann Hondo, IBM, Takeshi Imamura, IBM (Author), Mike Just, Entrust, Inc., Brian LaMacchia, Microsoft, Hiroshi Maruyama, IBM, John Messing, Law-on-Line, Shivaram Mysore, Sun Microsystems, Thane Plambeck, Verisign, Joseph Reagle, W3C (Chair, Editor), Aleksey Sanin, Jim Schaad, Soaring Hawk Consulting, Ed Simon, XMLsec (Author), Daniel Toth, Ford, Yongge Wang, Certicom, Steve Wiley, myProof.

Additionally, we thank the following for their comments during and subsequent to the Last Call of the original Recommendation: Martin Dürst, W3C, Dan Lanz, Zolera, Susan Lesch, W3C, David Orchard, BEA Systems, Ronald Rivest, MIT.

Contributions for version 1.1 were received from the members of the XML Security Working Group: Scott Cantor, Juan Carlos Cruellas, Pratik Datta, Gerald Edgar, Ken Graf, Phillip Hallam-Baker, Brad Hill, Frederick Hirsch, Brian LaMacchia, Konrad Lanz, Hal Lockhart, Cynthia Martin, Rob Miller, Sean Mullan, Shivaram Mysore, Magnus Nyström, Bruce Rich, Thomas Roessler, Ed Simon, Chris Solc, John Wray, Kelvin Yiu.

The working group also acknowledges the contribution of Juraj Somorovsky raising the issue of the CBC chosen ciphertext attack and contributions to revising the security considerations of XML Encryption 1.1.

2. Encryption Overview and Examples

This section is non-normative.

This section provides an overview and examples of XML Encryption syntax. The formal syntax is found in [section 3. Encryption Syntax](#) ; the specific processing is given in [Processing Rules](#) (section 4).

Expressed in shorthand form, the `EncryptedData` element has the following structure (where "?" denotes zero or one occurrence; "+" denotes one or more occurrences; "*" denotes zero or more occurrences; "|" denotes a choice; and the empty element tag means the element must be empty):

EXAMPLE 1

```
<EncryptedData Id? Type? MimeType? Encoding?>
  <EncryptionMethod/>?
  <ds:KeyInfo>
    <EncryptedKey>?
    <AgreementMethod>?
    <ds:KeyName>?
    <ds:RetrievalMethod>?
    <ds:*>?
  </ds:KeyInfo>?
  <CipherData>
    <CipherValue> | <CipherReference URI?>
  </CipherData>
  <EncryptionProperties>?
</EncryptedData>
```

The `CipherData` element envelopes or references the raw encrypted data. A `CipherData` element must have either a `CipherValue` or `CipherReference` child element. If enveloping, the raw encrypted data is the `CipherValue` element's content; if referencing, the `CipherReference` element's `URI` attribute points to the location of the raw encrypted data

2.1 Encryption Granularity

This section is non-normative.

Note: Examples in this document do not consider plaintext guessing attacks or other risks, and are only for illustrative purposes.

Consider the following fictitious payment information, which includes identification information and information appropriate to a payment method (e.g., credit card, money transfer, or electronic check):

EXAMPLE 2

```
<?xml version="1.0"?>

<PaymentInfo xmlns="http://example.org/paymentv2">
  <Name>John Smith</Name>
  <CreditCard Limit="5,000" Currency="USD">
    <Number>4019 2445 0277 5567</Number>
    <Issuer>Example Bank</Issuer>
    <Expiration>04/02</Expiration>
  </CreditCard>
</PaymentInfo>
```

This markup represents that John Smith is using his credit card with a limit of $5,000USD.

2.1.1 Encrypting an XML Element

This section is non-normative.

Smith's credit card number is sensitive information! If the application wishes to keep that information confidential, it can encrypt the `CreditCard` element:

EXAMPLE 3

```
<?xml version="1.0"?>
```

```
<PaymentInfo xmlns="http://example.org/paymentv2">
  <Name>John Smith</Name>
  <EncryptedData Type="http://www.w3.org/2001/04/xmlenc#Element"
                 xmlns="http://www.w3.org/2001/04/xmlenc#">
    <CipherData>
      <CipherValue>A23B45C56</CipherValue>
    </CipherData>
  </EncryptedData>
</PaymentInfo>
```

By encrypting the entire `CreditCard` element from its start to end tags, the identity of the element itself is hidden. (An eavesdropper doesn't know whether he used a credit card or money transfer.) The `CipherData` element contains the encrypted serialization of the `CreditCard` element.

2.1.2 Encrypting XML Element Content (Elements)

As an alternative scenario, it may be useful for intermediate agents to know that John used a credit card with a particular limit, but not the card's number, issuer, and expiration date. In this case, the content (character data or children elements) of the `CreditCard` element can be encrypted:

EXAMPLE 4

```
<?xml version="1.0"?>

<PaymentInfo xmlns="http://example.org/paymentv2">
  <Name>John Smith</Name>
  <CreditCard Limit="5,000" Currency="USD">
    <EncryptedData xmlns="http://www.w3.org/2001/04/xmlenc#"
                   Type="http://www.w3.org/2001/04/xmlenc#Content">
      <CipherData>
        <CipherValue>A23B45C56</CipherValue>
      </CipherData>
    </EncryptedData>
  </CreditCard>
</PaymentInfo>
```

2.1.3 Encrypting XML Element Content (Character Data)

Alternatively, consider the scenario in which all the information *except* the actual credit card number can be in the clear, including the fact that the Number element exists:

EXAMPLE 5

```
<?xml version="1.0"?>

<PaymentInfo xmlns="http://example.org/paymentv2">
  <Name>John Smith</Name>
  <CreditCard Limit="5,000" Currency="USD">
    <Number>
      <EncryptedData xmlns="http://www.w3.org/2001/04/xmlenc#"
                     Type="http://www.w3.org/2001/04/xmlenc#Content">
        <CipherData>
          <CipherValue>A23B45C56</CipherValue>
        </CipherData>
      </EncryptedData>
    </Number>
    <Issuer>Example Bank</Issuer>
    <Expiration>04/02</Expiration>
  </CreditCard>
</PaymentInfo>
```

Both `CreditCard` and `Number` are in the clear, but the character data content of `Number` is encrypted.

2.1.4 Encrypting Arbitrary Data and XML Documents

If the application scenario requires all of the information to be encrypted, the whole document is encrypted as an octet sequence. This applies to arbitrary data including XML documents.

EXAMPLE 6

```
<?xml version="1.0"?>

<EncryptedData xmlns="http://www.w3.org/2001/04/xmlenc#"
               MimeType="text/xml">
  <CipherData>
    <CipherValue>A23B45C56</CipherValue>
  </CipherData>
</EncryptedData>
```

Where appropriate, such as in the case of encrypting an entire EXI stream, the Type attribute SHOULD be provided and indicate the use of EXI. The optional MimeType MAY be used to record the actual (non-EXI-encoded) type, but is not necessary and may be omitted, as in the following EXI encryption example:

EXAMPLE 7

```
<?xml version="1.0"?>

<EncryptedData xmlns="http://www.w3.org/2001/04/xmlenc#"
```

```
                Type="http://www.w3.org/2009/xmlenc11#EXI">
  <CipherData>
    <CipherValue>A23B45C56</CipherValue>
  </CipherData>
</EncryptedData>
```

2.1.5 Super-Encryption: Encrypting EncryptedData

An XML document may contain zero or more `EncryptedData` elements. `EncryptedData` cannot be the parent or child of another `EncryptedData` element. However, the actual data encrypted can be anything, including `EncryptedData` and `EncryptedKey` elements (i.e., super-encryption). During super-encryption of an `EncryptedData` or `EncryptedKey` element, one must encrypt the entire element. Encrypting only the content of these elements, or encrypting selected child elements is an invalid instance under the provided schema.

For example, consider the following:

EXAMPLE 8

```
<pay:PaymentInfo xmlns:pay="http://example.org/paymentv2">
  <EncryptedData Id="ED1"
                 xmlns="http://www.w3.org/2001/04/xmlenc#"
                 Type="http://www.w3.org/2001/04/xmlenc#Element">
    <CipherData>
      <CipherValue>originalEncryptedData</CipherValue>
    </CipherData>
  </EncryptedData>
</pay:PaymentInfo>
```

A valid super-encryption of "`//xenc:EncryptedData[@Id='ED1']`" would be:

EXAMPLE 9

```
<pay:PaymentInfo xmlns:pay="http://example.org/paymentv2">
  <EncryptedData Id="ED2"
                 xmlns="http://www.w3.org/2001/04/xmlenc#"
                 Type="http://www.w3.org/2001/04/xmlenc#Element">
    <CipherData>
      <CipherValue>newEncryptedData</CipherValue>
    </CipherData>
  </EncryptedData>
</pay:PaymentInfo>
```

where the `CipherValue` content of '`newEncryptedData`' is the base64 encoding of the encrypted octet sequence resulting from encrypting the `EncryptedData` element with `Id='ED1'`.

2.2 `EncryptedData` and `EncryptedKey` Usage

2.2.1 `EncryptedData` with Symmetric Key (`KeyName`)

EXAMPLE 10

```
[s01]<EncryptedData xmlns="http://www.w3.org/2001/04/xmlenc#"
                    Type="http://www.w3.org/2001/04/xmlenc#Element">
[s02]  <EncryptionMethod
         Algorithm="http://www.w3.org/2001/04/xmlenc#tripledes-cbc"/>
[s03]  <ds:KeyInfo xmlns:ds="http://www.w3.org/2000/09/xmldsig#">
[s04]    <ds:KeyName>John Smith</ds:KeyName>
[s05]  </ds:KeyInfo>
[s06]  <CipherData><CipherValue>DEADBEEF</CipherValue></CipherData>
[s07]</EncryptedData>
```

`[s1]` The type of data encrypted may be represented as an attribute value to aid in decryption and subsequent processing. In this case, the data encrypted was an 'element'. Other alternatives include 'content' of an element, or an external octet sequence which can also be identified via the `MimeType` and `Encoding` attributes.

`[s2]` This (3DES CBC) is a symmetric key cipher.

`[s4]` The symmetric key has an associated name "John Smith".

`[s6]` `CipherData` contains a `CipherValue`, which is a base64 encoded octet sequence. Alternately, it could contain a `CipherReference`, which is a URI reference along with transforms necessary to obtain the encrypted data as an octet sequence

2.2.2 `EncryptedKey` (`ReferenceList`, `ds:RetrievalMethod`, `CarriedKeyName`)

The following `EncryptedData` structure is very similar to the one above, except this time the key is referenced using a `ds:RetrievalMethod`:

EXAMPLE 11

```
[t01]<EncryptedData Id="ED"
                    xmlns="http://www.w3.org/2001/04/xmlenc#">
[t02]  <EncryptionMethod
         Algorithm="http://www.w3.org/2001/04/xmlenc#aes128-cbc"/>
[t03]  <ds:KeyInfo xmlns:ds="http://www.w3.org/2000/09/xmldsig#">
[t04]    <ds:RetrievalMethod URI="#EK"
                             Type="http://www.w3.org/2001/04/xmlenc#EncryptedKey"/>
[t05]    <ds:KeyName>Sally Doe</ds:KeyName>
[t06]  </ds:KeyInfo>
```

```
[t07]  <CipherData><CipherValue>DEADBEEF</CipherValue></CipherData>
[t08]</EncryptedData>
```

[t02] This (AES-128-CBC) is a symmetric key cipher.

[t04] ds:RetrievalMethod is used to indicate the location of a key with type xenc:EncryptedKey. The (AES) key is located at '#EK'.

[t05] ds:KeyName provides an alternative method of identifying the key needed to decrypt the CipherData. Either or both the ds:KeyName and ds:KeyRetrievalMethod could be used to identify the same key.

Within the same XML document, there existed an EncryptedKey structure that was referenced within [t04]:

EXAMPLE 12

```
[t09]<EncryptedKey Id="EK" xmlns="http://www.w3.org/2001/04/xmlenc#">
[t10]  <EncryptionMethod
         Algorithm="http://www.w3.org/2001/04/xmlenc#rsa-1_5"/>
[t11]  <ds:KeyInfo xmlns:ds="http://www.w3.org/2000/09/xmldsig#">
[t12]    <ds:KeyName>John Smith</ds:KeyName>
[t13]  </ds:KeyInfo>
[t14]  <CipherData><CipherValue>xyzabc</CipherValue></CipherData>
[t15]  <ReferenceList>
[t16]    <DataReference URI="#ED"/>
[t17]  </ReferenceList>
[t18]  <CarriedKeyName>Sally Doe</CarriedKeyName>
[t19]</EncryptedKey>
```

[t09] The EncryptedKey element is similar to the EncryptedData element except that the data encrypted is always a key value.

[t10] The EncryptionMethod is the RSA public key algorithm.

[t12] ds:KeyName of "John Smith" is a property of the key necessary for decrypting (using RSA) the CipherData.

[t14] The CipherData's CipherValue is an octet sequence that is processed (serialized, encrypted, and encoded) by a referring encrypted object's EncryptionMethod. (Note, an EncryptedKey's EncryptionMethod is the algorithm used to encrypt these octets and does not speak about what type of octets they are.)

[t15-17] A ReferenceList identifies the encrypted objects (DataReference and KeyReference) encrypted with this key. The ReferenceList contains a list of references to data encrypted by the symmetric key carried within this structure.

[t18] The CarriedKeyName element is used to identify the encrypted key value which may be referenced by the KeyName element in ds:KeyInfo. (Since ID attribute values must be unique to a document,CarriedKeyName can indicate that several EncryptedKey structures contain the same key value encrypted for different recipients.)

3. Encryption Syntax

This section provides a detailed description of the syntax and features for XML Encryption. Features described in this section MUST be implemented unless otherwise noted. The syntax is defined via [XMLSCHEMA-1], [XMLSCHEMA-2] with the following XML preamble, declaration, internal entity, and import:

```
Schema Definition:

<?xml version="1.0" encoding="utf-8"?>
<!DOCTYPE schema  PUBLIC "-//W3C//DTD XMLSchema 200102//EN"
[
<!ATTLIST schema
xmlns:xenc CDATA #FIXED 'http://www.w3.org/2001/04/xmlenc'#
xmlns:ds CDATA #FIXED 'http://www.w3.org/2000/09/xmldsig#'> <!ENTITY xenc 'http://www.w3.org/2001/04/xmlenc#'> <! ENTIDAD % p


]>

<esquema xmlns = "http://www.w3.org/2001/XMLSchema" versión = "1.0" xmlns:ds = "http://www.w3.org/2000/09/xmldsig#" xmlns:xenc



  <importar espacio de nombres = "http://www.w3.org/2000/09/xmldsig#" esquemaLocation = "http://www.w3.org/TR/2002/

           REC-xmldsig-core-20020212/xmldsig-core-schema.xsd" />
```

(Nota: se agregó una nueva línea al URI de ubicación de esquema para que quepa en esta página, pero no forma parte del URI).

El marcado adicional definido en esta especificación utiliza el xenc11:espacio de nombres. La sintaxis se define en un esquema XML con el siguiente preámbulo:

```
Definición del esquema :

<? versión xml = "1.0" codificación = "utf-8" ?> <!DOCTYPE esquema PUBLIC "-//W3C//DTD XMLSchema 200102//EN"

[
<!ATTLIST esquema
xmlns:xenc CDATA #FIXED "http://www.w3.org/2001/04/xmlenc#"
xmlns:ds CDATA #FIXED "http://www.w3.org/2000/09/xmldsig#"
xmlns:xenc11 CDATA #FIXED "http://www.w3.org/2009/xmlenc11#"> <!ENTITY xenc "http://www.w3.org/2001/04/xmlenc#"> <!ENTITY % p


]>

<esquema xmlns = "http://www.w3.org/2001/XMLSchema" versión = "1.0" xmlns:xenc = "http://www.w3.org/2001/04/xmlenc#" xmlns:xen
```

```
<importar espacio de nombres = "http://www.w3.org/2000/09/xmldsig#" esquemaLocation = "http://www.w3.org/TR/2002/
                                REC-xmldsig-core-20020212/xmldsig-core-schema.xsd" />
<importar espacio de nombres = "http://www.w3.org/2001/04/xmlenc#" esquemaLocation = "http://www.w3.org/TR/2002/
                                REC-xmlenc-core-20021210/xenc-schema.xsd" />
```

(Nota: se agregó una nueva línea al URI de ubicación de esquema para que quepa en esta página, pero no forma parte del URI).

3.1 El EncryptedTypeelemento

EncryptedTypees el tipo abstracto del que se derivan EncryptedDatay . EncryptedKeySi bien estos dos últimos tipos de elementos son muy similares con respecto a sus modelos de contenido, una distinción sintáctica es útil para el procesamiento. Las implementaciones DEBEN generar un esquema laxamente válido [XMLSCHEMA-1], [XMLSCHEMA-2] EncryptedDatao EncryptedKeyelementos según lo especificado en las declaraciones de esquema posteriores. (Tenga en cuenta que la generación válida del esquema laxo significa que el contenido permitido xsd:ANYno necesita ser válido). Las implementaciones DEBEN crear estas estructuras XML (EncryptedTypeelementos y sus descendientes/contenido) en el Formulario de normalización C [NFC].

```
Definición del esquema :

<complexType name="EncryptedType" abstract="true">
  <sequence>
    <element name="EncryptionMethod" type="xenc:EncryptionMethodType"
             minOccurs="0"/>
    <element ref="ds:KeyInfo" minOccurs="0"/>
    <element ref="xenc:CipherData"/>
    <element ref="xenc:EncryptionProperties" minOccurs="0"/>
  </sequence>
  <attribute name="Id" type="ID" use="optional"/>
  <attribute name="Type" type="anyURI" use="optional"/>
  <attribute name="MimeType" type="string" use="optional"/>
  <attribute name="Encoding" type="anyURI" use="optional"/>
</complexType>
```

EncryptionMethod is an optional element that describes the encryption algorithm applied to the cipher data. If the element is absent, the encryption algorithm must be known by the recipient or the decryption will fail.

ds:KeyInfo is an optional element, defined by [XMLDSIG-CORE1], that carries information about the key used to encrypt the data. Subsequent sections of this specification define new elements that may appear as children of ds:KeyInfo.

CipherData is a mandatory element that contains the CipherValue or CipherReference with the encrypted data.

EncryptionProperties can contain additional information concerning the generation of the EncryptedType (e.g., date/time stamp).

Id is an optional attribute providing for the standard method of assigning a string id to the element within the document context.

Type is an optional attribute identifying type information about the plaintext form of the encrypted content. While optional, this specification takes advantage of it for processing described in section 4.4 Decryption. If the EncryptedData element contains data of Type 'element' or element 'content', and replaces that data in an XML document context, or contains data of Type 'EXI', it is strongly recommended the Type attribute be provided. Without this information, the decryptor will be unable to automatically restore the XML document to its original cleartext form.

MimeType is an optional (advisory) attribute which describes the media type of the data which has been encrypted. The value of this attribute is a string with values defined by [RFC2045]. For example, if the data that is encrypted is a base64 encoded PNG, the transfer Encoding may be specified as 'http://www.w3.org/2000/09/xmldsig#base64' and the MimeType as 'image/png'. This attribute is purely advisory; no validation of the MimeType information is required and it does not indicate the encryption application must do any additional processing. Note, this information may not be necessary if it is already bound to the identifier in the Type attribute. For example, the Element and Content types defined in this specification are always UTF-8 encoded text. In the case of Type EXI the MimeType attribute is not necessary, but if used should reflect the underlying type and not "EXI".

Encoding is an optional (advisory) attribute which describes the transfer encoding of the data that has been encrypted.

3.2 The EncryptionMethod Element

EncryptionMethod is an optional element that describes the encryption algorithm applied to the cipher data. If the element is absent, the encryption algorithm must be known to the recipient or the decryption will fail.

```
Schema Definition:

<complexType name="Encryptionmethodtype" mixed="true">
  <sequence>
    <element name="KeySize" minOccurs="0" type="xenc:KeySizeType"/>
    <element name="OAEPparams" minOccurs="0" type="base64Binary"/>
    <any namespace="##other" minOccurs="0" maxOccurs="unbounded"/>
  </sequence>
  <attribute name="Algorithm" type="anyURI" use="required"/>
</complexType>
```

The permitted child elements of the EncryptionMethod are determined by the specific value of the Algorithm attribute URI, and the KeySize child element is always permitted. For example, the RSA-OAEP algorithm (section 5.5.2 RSA-OAEP) uses the ds:DigestMethod and OAEPparams elements, and may use the xenc11:MGF element when needed. (We rely upon the ANY schema construct because it is not possible to specify element content based on the value of an attribute.)

The presence of any child element under EncryptionMethod that is not permitted by the algorithm or the presence of a KeySize child inconsistent with the algorithm MUST be treated as an error. (All algorithm URIs specified in this document imply a key size but this is not true in general. Most popular stream cipher algorithms take variable size keys.)

3.3 The `CipherData` Element

The `CipherData` is a mandatory element that provides the encrypted data. It must either contain the encrypted octet sequence as base64 encoded text as element content of the `CipherValue` element, or provide a reference to an external location containing the encrypted octet sequence via the `CipherReference` element.

```
Schema Definition:

<element name="CipherData" type="xenc:CipherDataType"/>

<complexType name="CipherDataType">
  <choice>
    <element name="CipherValue" type="base64Binary"/>
    <element ref="xenc:CipherReference"/>
  </choice>
</complexType>
```

3.3.1 The `CipherReference` Element

If `CipherValue` is not supplied directly, the `CipherReference` identifies a source which, when processed, yields the encrypted octet sequence.

The actual value is obtained as follows. The `CipherReference URI` contains an identifier that is dereferenced. Should the `CipherReference` element contain an OPTIONAL sequence of `Transform`s, the data resulting from dereferencing the URI is transformed as specified so as to yield the intended cipher value. For example, if the value is base64 encoded within an XML document; the transforms could specify an XPath expression followed by a base64 decoding so as to extract the octets.

The syntax of the URI and Transforms is defined in XML Signature [XMLDSIG-CORE1], however XML Encryption places the `Transforms` element in the XML Encryption namespace since it is used in XML Encryption to obtain an octet stream for decryption. In [XMLDSIG-CORE1] both generation and validation processing start with the same source data and perform that transform in the same order. In encryption, the decryptor has only the cipher data and the specified transforms are enumerated for the decryptor, in the order necessary to obtain the octets. Consequently, because it has different semantics `Transforms` is in the `xenc:` namespace.

For example, if the relevant cipher value is captured within a `CipherValue` element within a different XML document, the `CipherReference` might look as follows:

EXAMPLE 13

```
<CipherReference URI="http://www.example.com/CipherValues.xml">
  <Transforms>
    <ds:Transform Algorithm="http://www.w3.org/TR/1999/REC-xpath-19991116">
      <ds:XPath xmlns:xenc="http://www.w3.org/2001/04/xmlenc#">
        self::text()[parent::enc:CipherValue[@Id="example1"]]
      </ds:XPath> </ds:Transform> <ds:Transform Algorithm = "http://www.w3.org/2000/09/xmldsig#base64" /> </Transforms> </Ci
```

Las implementaciones DEBEN admitir la `CipherReference`función y la misma codificación URI, desreferenciación, esquema y códigos de respuesta HTTP que los de [XMLDSIG-CORE1]. La `Transform`característica y los algoritmos de transformación particulares son OPCIONALES .

```
Definición del esquema :

< nombre del elemento = "CipherReference" tipo = "xenc:CipherReferenceType" />

<complexType nombre = "CipherReferenceType" > <secuencia> <elemento nombre = "Transforms" tipo = "xenc:TransformsType" minOccu

<complexType nombre = "TransformsType" > <secuencia> <elemento ref = "ds:Transform" maxOccurs = "ilimitado" /> </secuencia> </
```

3.4 El `EncryptedData`elemento

El `EncryptedData`elemento es el elemento central de la sintaxis. Su hijo no solo `CipherData`contiene los datos cifrados, sino que también es el elemento que reemplaza el elemento cifrado, o el contenido del elemento, o sirve como la nueva raíz del documento.

```
Definición del esquema :

< nombre del elemento = "EncryptedData" tipo = "xenc:EncryptedDataType" />

<complexType nombre = "EncryptedDataType" > <complexContent> <extensión base = "xenc:EncryptedType" > </ extensión> </complexC
```

3.5 Extensiones al `ds:KeyInfo`elemento

Hay tres formas de `CipherData`proporcionar el material de claves necesario para descifrar:

1. El elemento `EncryptedData`o `EncryptedKey`especifica el material de clave asociado a través de un elemento secundario de `ds:KeyInfo`. Todos los elementos secundarios de ds: `KeyInfo`especificados en [XMLDSIG-CORE1] PUEDEN usarse como calificados:
    1. La compatibilidad con `ds:KeyValue`es OPCIONAL y se puede utilizar para transportar claves públicas, como valores clave Diffie-Hellman (sección 5.6.1 Valores clave Diffie-Hellman). (OBVIAMENTE NO SE RECOMIENDA incluir la clave de descifrado de texto sin formato, ya sea una clave privada o secreta).

2. Se RECOMIENDA el soporte de `ds:KeyName`para hacer referencia a un .`EncryptedKey CarriedKeyName`
3. `ds:RetrievalMethod`Se REQUIERE soporte para el mismo documento .

   Además, proporcionamos dos elementos secundarios adicionales: las aplicaciones DEBEN ser compatibles `EncryptedKey` ([sección 3.5.1 El elemento EncryptedKey](#)) y PUEDEN ser compatibles `AgreementMethod` ([sección 5.6 Acuerdo de clave](#)).
2. Un elemento separado (no dentro `ds:KeyInfo`) `EncryptedKey`puede especificar el `EncryptedData`o `EncryptedKey`al cual se aplicará su clave descifrada a través de un `DataReference`o `KeyReference`([sección 3.6 El elemento ReferenceList](#)).
3. El material de claves lo puede determinar el destinatario según el contexto de la aplicación y, por lo tanto, no es necesario mencionarlo explícitamente en el XML transmitido.

3.5.1 El `EncryptedKey`elemento

Identificador

`Type="http://www.w3.org/2001/04/xmlenc#EncryptedKey"`

(Esto se puede usar dentro de un `ds:RetrievalMethod`elemento para identificar el tipo de referente).

El `EncryptedKey`elemento se utiliza para transportar claves de cifrado desde el origen hasta uno o varios destinatarios conocidos. Puede usarse como un documento XML independiente, colocarse dentro de un documento de aplicación o aparecer dentro de un `EncryptedData`elemento como hijo de un `ds:KeyInfo` elemento. El valor de la clave siempre está cifrado para los destinatarios. Cuando `EncryptedKey`se descifra, los octetos resultantes se ponen a disposición del `EncryptionMethod`algoritmo sin ningún procesamiento adicional.

```
Definición del esquema :

< nombre del elemento = "EncryptedKey" tipo = "xenc:EncryptedKeyType" />

<complexType nombre = "EncryptedKeyType" > <complexContent> <extensión base = "xenc:EncryptedType" > <secuencia> <elemento ref
```

`ReferenceList`es un elemento opcional que contiene punteros a datos y claves cifradas con esta clave. La lista de referencias puede contener múltiples referencias `EncryptedKey`y `EncryptedData`elementos. Esto se hace usando elementos `KeyReference`y `DataReference`respectivamente. Estos se definen a continuación.

`CarriedKeyName`es un elemento opcional para asociar un nombre legible por el usuario con el valor clave. Esto luego se puede usar para hacer referencia a la clave usando el `ds:KeyName`elemento dentro de `ds:KeyInfo`. La misma `CarriedKeyName`etiqueta, a diferencia de un tipo de identificación, puede aparecer varias veces dentro de un solo documento. El valor de la clave DEBE ser el mismo en todos `EncryptedKey`los elementos identificados con la misma `CarriedKeyName`etiqueta dentro de un único documento XML. Tenga en cuenta que debido a que los espacios en blanco son significativos en el valor del `ds:KeyName` elemento, los espacios en blanco también son significativos en el valor del `CarriedKeyName`elemento.

`Recipient`es un atributo opcional que contiene una pista sobre a qué destinatario está destinado este valor de clave cifrada. Su contenido depende de la aplicación.

El `Type`atributo heredado de `EncryptedType` se puede utilizar para especificar aún más el tipo de clave cifrada si `EncryptionMethod Algorithm`no define una codificación/representación inequívoca. (Tenga en cuenta que todos los algoritmos de esta especificación tienen una representación inequívoca para sus estructuras clave asociadas).

3.5.2 El `DerivedKey`elemento

Identificador

`Type="http://www.w3.org/2009/xmlenc11#DerivedKey"`

(Esto se puede usar dentro de un `ds:RetrievalMethod`elemento para identificar el tipo de referente).

El `DerivedKey`elemento se utiliza para transportar información sobre una clave derivada desde el origen hasta los destinatarios. Puede usarse como un documento XML independiente, colocarse dentro de un documento de aplicación o aparecer dentro de un elemento `EncryptedData`o `Signature`como hijo de un `ds:KeyInfo` elemento. El valor clave en sí nunca lo envía el autor. Más bien, el originador proporciona información a los destinatarios mediante la cual los destinatarios pueden obtener el mismo valor clave. Cuando se ha obtenido la clave, los octetos resultantes se ponen a disposición del algoritmo `EncryptionMethod`o `SignatureMethod`sin ningún procesamiento adicional.

```
Definición del esquema :

<!-- targetNamespace='http://www.w3.org/2009/xmlenc11#' -->

< nombre del elemento = "DerivedKey" tipo = "xenc11:DerivedKeyType" />

<complexType nombre = "DerivedKeyType" > <secuencia> <elemento ref = "xenc11:KeyDerivationMethod" minOccurs = "0" /> <elementc
```

```
< nombre del elemento = "KeyDerivationMethod" tipo = "xenc:KeyDerivationMethodType" />

<complexType nombre = "KeyDerivationMethodType" > <secuencia> <cualquier espacio de nombres = "##cualquier" minOccurs = "0" ma
```

`KeyDerivationMethod`es un elemento opcional que describe el algoritmo de derivación de claves aplicado al material de clave maestra (subyacente). Si el elemento está ausente, el destinatario debe conocer el algoritmo de derivación de claves o la derivación de claves del destinatario fallará.

`ReferenceList`es un elemento opcional que contiene punteros a datos y claves cifradas con esta clave. La lista de referencias puede contener múltiples referencias `EncryptedKey`o `EncryptedData`elementos. Esto se hace utilizando `KeyReference`elementos `DataReference`de XML Encryption.

El elemento opcional `DerivedKeyName`se utiliza para identificar el valor clave derivado. Este elemento puede entonces ser referenciado por el `ds:KeyName`elemento en `ds:KeyInfo`. La misma `DerivedKeyName`etiqueta, a diferencia de un tipo de identificación, puede aparecer varias veces dentro de un solo documento. Tenga en cuenta que debido a que los espacios en blanco son significativos en el valor del `ds:KeyName`elemento, los espacios en blanco también son significativos en el valor del `DerivedKeyName`elemento.

`MasterKeyName`es un elemento opcional para asociar un nombre legible por el usuario con el valor de la clave maestra (o secreta). La misma `MasterKeyName`etiqueta, a diferencia de un tipo de identificación, puede aparecer varias veces dentro de un solo documento. El valor de la clave maestra DEBE ser el mismo en todos `DerivedKey`los elementos identificados con la misma `MasterKeyName`etiqueta dentro de un único documento XML. Si no `MasterKeyName`se proporciona, el destinatario debe conocer el material de la clave maestra o la derivación de la clave fallará.

`Recipient`es un atributo opcional que contiene una pista sobre a qué destinatario está destinado este valor de clave derivada. Su contenido depende de la aplicación.

El `Id`atributo opcional proporciona el método estándar para asignar una identificación de cadena al elemento dentro del contexto del documento.

El `Type`atributo se puede utilizar para especificar aún más el tipo de clave derivada si el `KeyDerivationMethod` algoritmo no define una codificación/representación inequívoca.

3.5.3 El `ds:RetrievalMethod`elemento

El con a de ' ' proporciona una manera de expresar un enlace a un elemento que contiene la clave necesaria para descifrar el elemento asociado con un o . El con a de ' ' proporciona una forma de expresar un vínculo a un elemento utilizado para derivar la clave necesaria para descifrar el elemento asociado con un o . El con uno de estos tipos siempre es hijo del elemento y puede aparecer varias veces. Si hay más de una instancia de a de este tipo, entonces los objetos a los que se hace referencia deben contener el mismo valor de clave, posiblemente cifrado de diferentes maneras o para diferentes destinatarios.`ds:RetrievalMethod` [XMLDSIG-CORE1]`Typehttp://www.w3.org/2001/04/xmlenc#EncryptedKeyEncryptedKeyCipherDataEncryptedDataEncryptedKeyds:RetrievalMethod` [XMLDSIG-CORE1]`Typehttp://www.w3.org/2001/04/xmlenc#DerivedKeyDerivedKeyCipherDataEncryptedDataEncryptedKeyds:RetrievalMethodds:KeyInfods:`

3.6 El `ReferenceList`elemento

`ReferenceList`es un elemento que contiene punteros desde un valor clave de `EncryptedKey`o `DerivedKey`a elementos cifrados por ese valor clave (`EncryptedData`o `EncryptedKey`elementos).

```
Definición del esquema :
< nombre del elemento = "ReferenceList" > <tipocomplejo> <elección minOccurs = "1" maxOccurs = "ilimitado" > < nombre del elem

<complexType nombre = "Tipo de referencia" > <secuencia> <cualquier espacio de nombres = "##otro" minOccurs = "0" maxOccurs =
```

`DataReference`Los elementos se utilizan para referirse a `EncryptedData`elementos que se cifraron utilizando la clave definida en el elemento `EncryptedKey`adjunto `DerivedKey`. Pueden aparecer varios `DataReference`elementos si `EncryptedData`existen varios elementos cifrados con la misma clave.

`KeyReference`Los elementos se utilizan para referirse a `EncryptedKey`elementos que se cifraron utilizando la clave definida en el elemento `EncryptedKey`adjunto `DerivedKey`. Pueden aparecer varios `KeyReference`elementos si `EncryptedKey`existen varios elementos cifrados con la misma clave.

Para ambos tipos de referencias, se pueden especificar opcionalmente elementos secundarios para ayudar al destinatario a recuperar los elementos `EncryptedKey`y/o `EncryptedData`. Estos podrían incluir información como transformaciones XPath, transformaciones de descompresión o información sobre cómo recuperar los elementos de una instalación de almacenamiento de documentos. Por ejemplo:

EJEMPLO 14

```
<ReferenceList> <DataReference URI = "#invoice34" > <ds:Transforms> <ds:Transform Algorithm = "http://www.w3.org/TR/1999/REC

          self::xenc:EncryptedData[@Id="ejemplo1"]
        </ds:XPath> </ds:Transform> </ds:Transforms> </DataReference> </ReferenceList>
```

3.7 El `EncryptionProperties`elemento

Identificador

`Type="http://www.w3.org/2001/04/xmlenc#EncryptionProperties"`

(Esto se puede usar dentro de un `ds:Reference`elemento para identificar el tipo de referente).

Se pueden colocar elementos de información adicional relacionados con la generación de `EncryptedData`o en un elemento (por ejemplo, marca de fecha/hora o el número de serie del hardware criptográfico utilizado durante el cifrado). El atributo identifica la estructura que se describe. permite la inclusión de atributos del espacio de nombres XML que se incluirán (es decir, , y).`EncryptedKeyEncryptionPropertyTargetEncryptedTypeanyAttributexml:spacexml:langxml:base`

```
Definición del esquema :

< nombre del elemento = "EncryptionProperties" tipo = "xenc:EncryptionPropertiesType" />

<complexType nombre = "EncryptionPropertiesType" > <secuencia> <elemento ref = "xenc:EncryptionProperty" maxOccurs = "ilimitad

< nombre del elemento = "EncryptionProperty" tipo = "xenc:EncryptionPropertyType" />

<complexType nombre = "EncryptionPropertyType" mixto = "verdadero" > <choice maxOccurs = "ilimitado" > <cualquier espacio de n
```

4. Reglas de procesamiento

Esta sección describe las operaciones que se realizarán como parte del procesamiento de cifrado y descifrado mediante implementaciones de esta especificación. Los requisitos de conformidad se especifican en los siguientes roles:

cifrador
: Una implementación de cifrado XML con la función de cifrar datos.

Descifrador
: Una implementación de cifrado XML con la función de descifrar datos.

La aplicación invoca Encryptor y Decryptor . Esta especificación no incluye definiciones normativas para el comportamiento de las aplicaciones. Sin embargo, esta especificación incluye requisitos de conformidad sobre datos cifrados que solo pueden lograrse mediante un comportamiento apropiado por parte de las tres partes. Depende de los contextos de implementación específicos cómo se logra esto.

4.1 Modelo de aplicación previsto

Las reglas de procesamiento para el cifrado XML están diseñadas en torno a un modelo de aplicación previsto que esta versión de la especificación no cubre normativamente.

En el modelo de procesamiento previsto, el cifrado XML se utiliza para cifrar una secuencia de octetos, una secuencia EXI o un fragmento de un documento XML que coincide con la `content`producción `element` de [XML10].

Si se utiliza el cifrado XML con algún flujo de octetos, la codificación y el significado precisos de ese flujo de octetos dependen de la aplicación, pero el cifrador o el descifrador lo tratan como opaco. La aplicación puede utilizar los parámetros `Type`, `Encoding`y `MimeType`para transportar más información sobre la naturaleza de ese flujo de octetos. Por lo tanto, un `Type`parámetro desconocido, en general, no es tratado como un error ni por el cifrador ni por el descifrador, sino que simplemente lo pasa junto con los demás parámetros pertinentes y el flujo de octetos de texto sin cifrar.

Si se utiliza el cifrado XML con XML `element`o XML `content`, los cifradores y descifradores suelen realizar un procesamiento específico del tipo:

- Si un `element`está cifrado, entonces Encryptor reemplazará el elemento en cuestión con un `EncryptedData`elemento construido apropiadamente. El Decryptor, por el contrario, reemplazará el `EncryptedData`elemento con su texto sin cifrar.
- Si XML `content`está cifrado, el Encryptor también reemplazará este contenido con un `EncryptedData`elemento construido apropiadamente y el Decryptor revertirá esta operación.

Tenga en cuenta que el comportamiento previsto de Encryptor a menudo hará que el documento con partes cifradas deje de ser válido con respecto a su esquema para el formato XML de alojamiento, a menos que ese formato esté específicamente preparado para usarse con XML Encryption. NO SE REQUIERE un cifrador o descifrador que implemente el modelo de procesamiento previsto para garantizar que el XML resultante tenga un esquema válido para el formato XML de alojamiento.

Si el procesamiento XML se maneja dentro de Encryptor y Decryptor, y se utilizan los `Type`valores de atributos para `element`y texto sin cifrar, entonces Encryptor y Decryptor DEBEN garantizar que el texto sin cifrar XML se serialice como UTF-8 antes del cifrado y, si es necesario, se vuelva a convertir. a cualquier otra codificación que pueda ser utilizada por el contexto XML circundante.`content`

Si se utiliza el cifrado XML con una secuencia EXI [EXI], los cifradores y descifradores procesan el contenido como para el procesamiento de elementos XML o contenido XML, pero teniendo en cuenta la serialización EXI. En particular, el cifrador reemplazará el elemento XML o el fragmento XML en cuestión con un elemento EncryptedData construido adecuadamente. Por el contrario, Decryptor reemplazará el elemento EncryptedData con su elemento XML de texto sin formato o fragmento XML. Tenga en cuenta que el documento XML en el que

está incrustado el elemento EncryptedData puede codificarse usando EXI y/o EXI puede usarse para codificar el texto sin cifrar antes del cifrado.

4.2`Type` Valores de parámetros conocidos

PARA FINES DE INTEROPERABILIDAD, DEBEN implementarse los siguientes tipos de modo que una implementación pueda tomar como entrada y producir como salida datos que coincidan con las reglas de producción 39 y 43 de [XML10]:

elemento ' **http://www.w3.org/2001/04/xmlenc#Element** '
"[39] elemento ::= `EmptyElemTag` | Contenido de STag ETag "
contenido ' **http://www.w3.org/2001/04/xmlenc#Content** '
"[43] contenido ::= `CharData`? ((elemento | Referencia | CDSect | PI | Comentario) `CharData`?)*"

La compatibilidad con el siguiente tipo es OPCIONAL para Encryptors y Decryptors:

http://www.w3.org/2009/xmlenc11#EXI
La presencia de esto `Type`indica que el texto sin cifrar es una secuencia EXI [EXI]. Los cifradores y descifradores que admiten este tipo PUEDEN operar directamente en (partes de) flujos EXI.

Los cifradores y descifradores DEBEN manejar valores de atributos desconocidos o vacíos `Type`como señal de que el texto sin cifrar debe manejarse como un flujo de octetos opaco, cuyo procesamiento específico depende de la aplicación que lo invoca. En este caso, los parámetros `Type`, `MimeType`y DEBEN tratarse como datos opacos cuyo procesamiento apropiado depende de la aplicación.`Encoding`

4.3 Cifrado

La selección del algoritmo, los parámetros y las claves de cifrado está fuera del alcance de esta especificación.

Se supone que los datos en texto claro están presentes como un tren de octetos. Si el texto sin formato es de tipo `elemento` o `content`, los datos DEBEN serializarse en UTF-8 como se especifica en [XML10], utilizando la forma normal C [NFC].

Para que cada elemento de datos se cifre como elemento `EncryptedData`o `EncryptedKey`, el **cifrador** DEBE :

1. Obtener (o derivar) y (opcionalmente) representar la clave.
    1. Si la clave debe identificarse (mediante nombre, URI o incluirse en un elemento secundario), construya la clave `ds:KeyInfo`según corresponda (p. ej. `ds:KeyName`, `ds:KeyValue`, `ds:RetrievalMethod`, etc.)
    2. Si se va a cifrar la clave en sí, construya un `EncryptedKey`elemento aplicando recursivamente este proceso de cifrado. El resultado puede ser hijo de `ds:KeyInfo`o puede existir en otro lugar y puede identificarse en el paso anterior.
    3. If the key was derived from a master key, construct a `DerivedKey` element with associated child elements. The result may, as in the `EncryptedKey` case, be a child of `ds:KeyInfo`, or it may exist elsewhere.
2. Encrypt the data:
    1. Encrypt the octets using the algorithm and key.
    2. Unless the **decryptor** will implicitly know the type of the encrypted data, the **encryptor** SHOULD set the `Type` to indicate the intended interpretation of the cleartext data. See section 4.2 Well-known Type parameter values for known parameter values.

       If the data is a simple octet sequence it MAY be described with the `MimeType` and `Encoding` attributes. For example, the data might be an XML document (`MimeType="text/xml"`), sequence of characters (`MimeType="text/plain"`), or binary image data (`MimeType="image/png"`).
3. Build the `EncryptedData` or `EncryptedKey` structure:

   An `EncryptedData` or `EncryptedKey` structure represents all of the information previously discussed including the type of the encrypted data, encryption algorithm, parameters, key, type of the encrypted data, etc.
    1. If the encrypted octet sequence obtained in step 2 is to be stored in the `CipherData` element within the `EncryptedData` or `EncryptedKey` element, then the base64 representation of the encrypted octet sequence is inserted as the content of a `CipherValue` element.
    2. If the encrypted octet sequence is stored externally to the `EncryptedData` or `EncryptedKey` element, then the URI and transforms (if any) required for the Decryptor to retrieve the encrypted octet sequence are described within a `CipherReference` element.

4.4 Decryption

For each `EncryptedData` or `EncryptedKey` to be decrypted, the **decryptor** MUST:

1. Determine the algorithm, parameters and key information to be used. This information may be obtained out-of-band, or determined according to a `ds:KeyInfo` element; see section 3.5 Extensions to ds:KeyInfo Element.
2. Decrypt the data contained in the `CipherData` element.
    1. If a `CipherValue` child element is present, then the associated text value is retrieved and base64 decoded so as to obtain the encrypted octet sequence.
    2. If a `CipherReference` child element is present, the URI and transforms (if any) are used to retrieve the encrypted octet sequence.
    3. The encrypted octet sequence is decrypted using the algorithm, parameters and key value already determined from step 1.

4.5 XML Encryption

Encryption and decryption operations are operations on octets. The **application** is responsible for the marshalling XML such that it can be serialized into an octet sequence, encrypted, decrypted, and be of use to the recipient.

For example, if the application wishes to canonicalize its data or encode/compress the data in an XML packaging format, the application needs to marshal the XML accordingly and identify the resulting type via the `EncryptedData` `Type` attribute. The likelihood of successful decryption and subsequent processing will be dependent on the recipient's support for the given type. Also, if the data is intended to be processed both before encryption and after decryption (e.g., XML Signature [XMLDSIG-CORE1] validation or an XSLT transform) the encrypting application must be careful to preserve information necessary for that process's success.

The following sections contain specifications for decrypting, replacing, and serializing XML content (i.e., `Type` 'element' or element 'content') using the [XPATH] data model. These sections are non-normative and OPTIONAL to implementers of this specification, but they may be normatively referenced by and be required by other specifications that require a consistent processing for applications, such as [XMLENC-DECRYPT].

4.5.1 A Decrypt Implementation (Non-normative)

Where *P* is the context in which the serialized XML should be parsed (a document node or element node) and *O* is the octet sequence representing UTF-8 encoded characters resulting from step 4.3 in section 4.4 Decryption. *Y* is node-set representing the decrypted content obtained by the following steps:

1. Let *C* be the parsing context of a child of *P*, which consists of the following items:
   - Prefix and namespace name of each namespace that is in scope for *P*.
   - Name and value of each general entity that is effective for the XML document causing *P*.
2. Wrap the decrypted octet stream *O* in the context *C* as specified in section 4.5.4 Text Wrapping.
3. Parse the wrapped octet stream as described in The Reference Processing Model (section 4.3.3.2) of [XMLDSIG-CORE1], resulting in a node-set.
4. *Y* is the node-set obtained by removing the root node, the wrapping element node, and its associated set of attribute and namespace nodes from the node-set obtained in Step 3.

4.5.2 A Decrypt and Replace Implementation (Non-normative)

Where *X* is the [XPATH] node set corresponding to an XML document and *e* is an `EncryptedData` element node in *X*.

1. *Z* is an [XPATH] node-set that identical to X except where the element node *e* is an `EncryptedData` element type. In which case:
   1. Decrypt *e* in the context of its parent node as specified in the section 4.5.1 A Decrypt Implementation (Non-normative) yielding *Y*, an [XPATH] node set.
   2. Include *Y* in place of *e* and its descendants in *X*. Since [XPATH] does not define methods of replacing node-sets from different documents, the result MUST be equivalent to replacing e with the octet stream resulting from its decryption in the serialized form of *X* and re-parsing the document. However, the actual method of performing this operation is left to the implementor.

4.5.3 Serializing XML (Non-normative)

4.5.3.1 Default Namespace Considerations

In section 4.3 Encryption (step 3.1), when serializing an XML fragment special care SHOULD be taken with respect to default namespaces. If the data will be subsequently decrypted in the context of a parent XML document then serialization can produce elements in the wrong namespace. Consider the following fragment of XML:

EXAMPLE 15

```
<Document xmlns="http://example.org/">
  <ToBeEncrypted xmlns="" />
</Document>
```

Serialization of the element `ToBeEncrypted` fragment via [XML-C14N] would result in the characters "`<ToBeEncrypted></ToBeEncrypted>`" as an octet stream. The resulting encrypted document would be:

EXAMPLE 16

```
<Document xmlns="http://example.org/">
  <EncryptedData xmlns="...">
    <!-- Containing the encrypted "<ToBeEncrypted></ToBeEncrypted>" -->
  </EncryptedData>
</Document>
```

Decrypting and replacing the `EncryptedData` within this document would produce the following incorrect result:

EXAMPLE 17

```
<Document xmlns="http://example.org/">
  <ToBeEncrypted/>
</Document>
```

This problem arises because most XML serializations assume that the serialized data will be parsed directly in a context where there is no default namespace declaration. Consequently, they do not redundantly declare the empty default namespace with an `xmlns=""`. If, however, the serialized data is parsed in a context where a default namespace declaration is in scope (e.g., the parsing context as described in section 4.5.1 A Decrypt Implementation (Non-normative)), then it may affect the interpretation of the serialized data.

To solve this problem, a canonicalization algorithm MAY be augmented as follows for use as an XML encryption serializer:

- A default namespace declaration with an empty value (i.e., `xmlns=""`) SHOULD be emitted where it would normally be suppressed by the canonicalization algorithm.

While the result may not be in proper canonical form, this is harmless as the resulting octet stream will not be used directly in a [XMLDSIG-CORE1] signature value computation. Returning to the preceding example with our new augmentation, the `ToBeEncrypted` element would be serialized as follows:

```
<ToBeEncrypted xmlns=""></ToBeEncrypted>
```

When processed in the context of the parent document, this serialized fragment will be parsed and interpreted correctly.

This augmentation can be retroactively applied to an existing canonicalization implementation by canonicalizing each apex node and its descendants from the node set, inserting `xmlns=""` at the appropriate points, and concatenating the resulting octet streams.

4.5.3.2 XML Attribute Considerations

Similar attention between the relationship of a fragment and the context into which it is being inserted should be given to the `xml:base`, `xml:lang`, and `xml:space` attributes as mentioned in the Security Considerations of [XML-EXC-C14N]. For example, if the element:

EXAMPLE 18

```
<Bongo href="example.xml"/>
```

is taken from a context and serialized with no `xml:base` [XMLBASE] attribute and parsed in the context of the element:

EXAMPLE 19

```
<Baz xml:base="http://example.org/"/>
```

the result will be:

EXAMPLE 20

```
<Baz xml:base="http://example.org/"><Bongo href="example.xml"/></Baz>
```

`Bongo`'s `href` is subsequently interpreted as "`http://example.org/example.xml`". If this is not the correct URI, `Bongo` should have been serialized with its own `xml:base` attribute.

Unfortunately, the recommendation that an empty value be emitted to divorce the default namespace of the fragment from the context into which it is being inserted cannot be made for the attributes `xml:base`, and `xml:space`. (Error 41 of the XML 1.0 Second Edition Specification Errata clarifies that an empty string value of the attribute `xml:lang` *is* considered as if, "there is no language information available, just as if `xml:lang` had not been specified".)The interpretation of an empty value for the `xml:base` or `xml:space` attributes is undefined or maintains the contextual value. Consequently, applications SHOULD ensure (1) fragments that are to be encrypted are not dependent on XML attributes, or (2) if they are dependent and the resulting document is intended to be valid [XML10], the fragment's definition permits the presence of the attributes and that the attributes have non-empty values.

4.5.4 Text Wrapping

This section specifies the process for wrapping text in a given parsing context. The process is based on the proposal by Richard Tobin [Tobin] for constructing the infoset [XML-INFOSET] of an external entity.

The process consists of the following steps:

1. If the parsing context contains any general entities, then emit a document type declaration that provides entity declarations.
2. Emit a `dummy` element start-tag with namespace declaration attributes declaring all the namespaces in the parsing context.
3. Emit the text.
4. Emit a `dummy` element end-tag.

In the above steps, the document type declaration and `dummy` element tags MUST be encoded in UTF-8.

Consider the following document containing an `EncryptedData` element:

EXAMPLE 21

```
<!DOCTYPE Document [
<!ENTITY dsig "http://www.w3.org/2000/09/xmldsig#">
]>
<Document xmlns="http://example.org/">
  <foo:Body xmlns:foo="http://example.org/foo">
    <EncryptedData xmlns="http://www.w3.org/2001/04/xmlenc#"
                   Type="http://www.w3.org/2001/04/xmlenc#Element">
      ...
    </EncryptedData>
  </foo:Body>
</Document>
```

If the `EncryptedData` element is decrypted to the text "`<One><foo:Two/></One>`", then the wrapped form is as follows:

EXAMPLE 22

```
<!DOCTYPE dummy [
<!ENTITY dsig "http://www.w3.org/2000/09/xmldsig#">
]>
<dummy xmlns="http://example.org/"
       xmlns:foo="http://example.org/foo">
  <One>
    <foo:Two/>
  </One>
</dummy>
```

5. Algorithms

This section discusses algorithms used with the XML Encryption specification. Entries contain the identifier to be used as the value of the `Algorithm` attribute of the `EncryptionMethod` element or other element representing the role of the algorithm, a reference to the formal specification, definitions for the representation of keys and the results of cryptographic operations where applicable, and general applicability comments.

5.1 Algorithm Identifiers and Implementation Requirements

All algorithms listed below have implicit parameters depending on their role. For example, the data to be encrypted or decrypted, keying material, and direction of operation (encrypting or decrypting) for encryption algorithms. Any explicit additional parameters to an algorithm appear as content elements within the element. Such parameter child elements have descriptive element names, which are frequently algorithm specific, and SHOULD be in the same namespace as this XML Encryption specification, the XML Signature specification, or in an algorithm specific namespace. An example of such an explicit parameter could be a nonce (unique quantity) provided to a key agreement algorithm.

This specification defines a set of algorithms, their URIs, and requirements for implementation. Levels of requirement specified, such as "REQUIRED" or "OPTIONAL", refer to implementation, not use. Furthermore, the mechanism is extensible, and alternative algorithms may be used.

5.1.1 Table of Algorithms

The table below lists the categories of algorithms. Within each category, a brief name, the level of implementation requirement, and an identifying URI are given for each algorithm.

Block Encryption

1. REQUIRED TRIPLEDES
   http://www.w3.org/2001/04/xmlenc#tripledes-cbc
2. REQUIRED AES-128
   http://www.w3.org/2001/04/xmlenc#aes128-cbc
3. REQUIRED AES-256
   http://www.w3.org/2001/04/xmlenc#aes256-cbc
4. REQUIRED AES128-GCM
   http://www.w3.org/2009/xmlenc11#aes128-gcm
5. OPTIONAL AES-192
   http://www.w3.org/2001/04/xmlenc#aes192-cbc
6. OPTIONAL AES192-GCM
   http://www.w3.org/2009/xmlenc11#aes192-gcm
7. OPTIONAL AES256-GCM
   http://www.w3.org/2009/xmlenc11#aes256-gcm

Note: Use of AES GCM is strongly recommended over any CBC block encryption algorithms as recent advances in cryptanalysis [XMLENC-CBC-ATTACK][XMLENC-CBC-ATTACK-COUNTERMEASURES] have cast doubt on the ability of CBC block encryption algorithms to protect plain text when used with XML Encryption. Other mitigations should be considered when using CBC block encryption, such as conveying the encrypted data over a secure channel such as TLS. The CBC block encryption algorithms that are listed as required remain so for backward compatibility.

Stream Encryption

1. none
   Syntax and recommendations are given below to support user specified algorithms.

Key Derivation

1. REQUIRED ConcatKDF
   http://www.w3.org/2009/xmlenc11#ConcatKDF
2. OPTIONAL PBKDF2
   http://www.w3.org/2009/xmlenc11#pbkdf2

Key Transport

1. REQUIRED RSA-OAEP (including MGF1 with SHA1)
   http://www.w3.org/2001/04/xmlenc#rsa-oaep-mgf1p
2. Optional RSA-OAEP
   http://www.w3.org/2009/xmlenc11#rsa-oaep
3. OPTIONAL RSA-v1.5 (see RSA-v1.5 security note)
   http://www.w3.org/2001/04/xmlenc#rsa-1_5

Key Agreement

1. REQUIRED Elliptic Curve Diffie-Hellman (Ephemeral-Static mode)
   http://www.w3.org/2009/xmlenc11#ECDH-ES

2. OPTIONAL Diffie-Hellman Key Agreement (Ephemeral-Static mode) with Legacy Key Derivation Function
   http://www.w3.org/2001/04/xmlenc#dh
3. OPTIONAL Diffie-Hellman Key Agreement (Ephemeral-Static mode) with explicit Key Derivation Functions
   http://www.w3.org/2009/xmlenc11#dh-es

Symmetric Key Wrap

1. REQUIRED TRIPLEDES KeyWrap
   http://www.w3.org/2001/04/xmlenc#kw-tripledes
2. REQUIRED AES-128 KeyWrap
   http://www.w3.org/2001/04/xmlenc#kw-aes128
3. REQUIRED AES-256 KeyWrap
   http://www.w3.org/2001/04/xmlenc#kw-aes256
4. OPTIONAL AES-192 KeyWrap
   http://www.w3.org/2001/04/xmlenc#kw-aes192

Message Digest

1. REQUIRED SHA1 (*Use is DISCOURAGED*; see below).
   http://www.w3.org/2000/09/xmldsig#sha1
2. REQUIRED SHA256
   http://www.w3.org/2001/04/xmlenc#sha256
3. OPTIONAL SHA384
   http://www.w3.org/2001/04/xmlenc#sha384
4. OPTIONAL SHA512
   http://www.w3.org/2001/04/xmlenc#sha512
5. OPTIONAL RIPEMD-160
   http://www.w3.org/2001/04/xmlenc#ripemd160

Canonicalization

1. OPTIONAL Canonical XML 1.0 (omit comments)
   http://www.w3.org/TR/2001/REC-xml-c14n-20010315
2. OPTIONAL Canonical XML 1.0 (with comments)
   http://www.w3.org/TR/2001/REC-xml-c14n-20010315#WithComments
3. OPTIONAL Canonical XML 1.1 (omit comments)
   http://www.w3.org/2006/12/xml-c14n11
4. OPTIONAL Canonical XML 1.1 (with comments)
   http://www.w3.org/2006/12/xml-c14n11#WithComments
5. OPTIONAL Exclusive XML Canonicalization 1.0 (omit comments)
   http://www.w3.org/2001/10/xml-exc-c14n#
6. OPTIONAL Exclusive XML Canonicalization 1.0 (with comments)
   http://www.w3.org/2001/10/xml-exc-c14n#WithComments

Encoding

1. REQUIRED base64 (*note)
   http://www.w3.org/2000/09/xmldsig#base64

Transforms

1. REQUIRED base64 (*note)
   http://www.w3.org/2000/09/xmldsig#base64

*note: The same URI is used to identify base64 both in "encoding" context (e.g. when used with the `Encoding` attribute of an `EncryptedKey` element, see section 3.1 The EncryptedType Element) as well as in "transform" context (when identifying a base64 transform for a `CipherReference`, see section 3.3.1 The CipherReference Element).

5.2 Block Encryption Algorithms

Block encryption algorithms are designed for encrypting and decrypting data in fixed size, multiple octet blocks. Their identifiers appear as the value of the `Algorithm` attributes of `EncryptionMethod` elements that are children of `EncryptedData`.

Nota : Los algoritmos de cifrado de bloques CBC no deben utilizarse sin tener en cuenta posibles riesgos de seguridad graves .

Los algoritmos de cifrado de bloques toman, como argumentos implícitos, los datos que se van a cifrar o descifrar, el material de clave y su dirección de operación. Para todos estos algoritmos especificados a continuación, se requiere un vector de inicialización (IV) codificado con el texto cifrado. Para los algoritmos de cifrado de bloques especificados por el usuario, el IV, si lo hubiera, podría especificarse junto con los datos cifrados, como un elemento de contenido del algoritmo o en cualquier otro lugar.

El IV está codificado con y antes del texto cifrado para los algoritmos siguientes para facilitar la disponibilidad del código de descifrado y enfatizar su asociación con el texto cifrado. Las buenas prácticas criptográficas requieren que se utilice un IV diferente para cada cifrado.

5.2.1 Relleno

Dado que los datos que se cifran son un número arbitrario de octetos, es posible que no sean un múltiplo del tamaño del bloque. Esto se resuelve rellenando el texto sin formato hasta el tamaño del bloque antes del cifrado y deshaciendo el relleno después del descifrado. El algoritmo de relleno consiste en calcular el número de octetos más pequeño distinto de cero, por ejemplo `N`, que debe añadirse al texto sin formato para que sea un múltiplo del tamaño del bloque. Supondremos que el tamaño del bloque es `B` de octetos, por lo que `N`está en el rango de 1 a `B`. Rellene añadiendo al texto sin formato un sufijo de `N-1`bytes de relleno arbitrarios y un byte final cuyo valor sea `N`. Al descifrar, simplemente tome el último byte y, después de verificarlo, elimine esa cantidad de bytes del final del texto cifrado descifrado.

Por ejemplo, supongamos un tamaño de bloque de 8 bytes y un texto sin formato de `0x616263`. El texto sin formato acolchado estaría entonces `0x616263??????????05`donde "??" Los bytes pueden tener cualquier valor. De manera similar, el texto sin formato de `0x2122232425262728`se rellenaría con `0x2122232425262728??????????????08`.

5.2.2 Triple DES

Identificador:
http://www.w3.org/2001/04/xmlenc#tripledes-cbc

NIST SP800-67 [SP800-67] especifica tres operaciones FIPS 46-3 [DES] secuenciales. El cifrado XML TRIPLEDES consta de un cifrado DES, un descifrado DES y un cifrado DES utilizado en el modo Cipher Block Chaining (CBC) con 192 bits de clave y un vector de inicialización (IV) de 64 bits. De los bits clave, los primeros 64 bits se utilizan en la primera operación DES, los segundos 64 bits en la operación DES intermedia y los terceros 64 bits en la última operación DES.

Nota: Cada uno de estos 64 bits de clave contiene 56 bits efectivos y 8 bits de paridad. Por tanto, sólo hay 168 bits operativos de los 192 que se transportan para una clave TRIPLEDES. (Dependiendo del criterio utilizado para el análisis, se puede pensar que la fuerza efectiva de la clave es de 112 bits (debido a los ataques intermedios) o incluso menos).

El texto cifrado resultante tiene el prefijo IV. Si se incluye en la salida XML, está codificado en base64. Un ejemplo de método de cifrado TRIPLEDES es el siguiente:

EJEMPLO 23

```
< Algoritmo del método de cifrado = "http://www.w3.org/2001/04/xmlenc#tripledes-cbc" />
```

Nota : Los algoritmos de cifrado de bloques CBC no deben utilizarse sin tener en cuenta posibles riesgos de seguridad graves .

5.2.3 AES

Identificador:
http://www.w3.org/2001/04/xmlenc#aes128-cbc
http://www.w3.org/2001/04/xmlenc#aes192-cbc
http://www.w3.org/2001/04/xmlenc#aes256-cbc

[AES] se utiliza en el modo Cipher Block Chaining (CBC) con un vector de inicialización (IV) de 128 bits. El texto cifrado resultante tiene el prefijo IV. Si se incluye en la salida XML, está codificado en base64. Un ejemplo de método de cifrado AES es el siguiente:

EJEMPLO 24

```
< Algoritmo del método de cifrado = "http://www.w3.org/2001/04/xmlenc#aes128-cbc" />
```

Nota : Los algoritmos de cifrado de bloques CBC no deben utilizarse sin tener en cuenta posibles riesgos de seguridad graves .

5.2.4 AES-GCM

Identificador:
http://www.w3.org/2009/xmlenc11#aes128-gcm
http://www.w3.org/2009/xmlenc11#aes192-gcm
http://www.w3.org/2009/xmlenc11#aes256-gcm

AES-GCM [SP800-38D] es un mecanismo de cifrado autenticado. Equivale a realizar estas dos operaciones en un solo paso: cifrado AES seguido de firma HMAC.

AES-GCM es muy atractivo desde el punto de vista del rendimiento porque el costo de AES-GCM es similar al del cifrado AES-CBC normal, pero logra el mismo resultado que el cifrado y la firma HMAC. También se puede canalizar AES-GCM para que sea susceptible de aceleración por hardware.

A los efectos de esta especificación, AES-GCM se utilizará con un vector de inicialización (IV) de 96 bits y una etiqueta de autenticación (T) de 128 bits. El texto cifrado contiene primero el IV, seguido de los octetos cifrados y finalmente la etiqueta de autenticación. No se debe utilizar ningún relleno durante el cifrado. Durante el descifrado, la implementación debe comparar la etiqueta de autenticación calculada durante el descifrado con la etiqueta de autenticación especificada y fallar si no coinciden. Para obtener detalles sobre la implementación de AES-GCM, consulte [SP800-38D].

5.3 Algoritmos de cifrado de flujo

Simple stream encryption algorithms generate, based on the key, a stream of bytes which are XORed with the plain text data bytes to produce the cipher text on encryption and with the cipher text bytes to produce plain text on decryption. They are normally used for the encryption of data and are specified by the value of the `Algorithm` attribute of the `EncryptionMethod` child of an `EncryptedData` element.

NOTE: It is critical that each simple stream encryption key (or key and initialization vector (IV) if an IV is also used) be used once only. If the same key (or key and IV) is ever used on two messages then, by XORing the two cipher texts, you can obtain the XOR of the two plain texts. This is usually very compromising.

No specific stream encryption algorithms are specified herein but this section is included to provide general guidelines.

Stream algorithms typically use the optional `KeySize` explicit parameter. In cases where the key size is not apparent from the algorithm URI or key source, as in the use of key agreement methods, this parameter sets the key size. If the size of the key to be used is apparent and disagrees with the `KeySize` parameter, an error MUST be returned. Implementation of any stream algorithms is optional. The schema for the KeySize parameter is as follows:

```
Schema Definition:

<simpleType name="KeySizeType">
  <restriction base="integer"/>
</simpleType>
```

5.4 Key Derivation

Key derivation is a well-established mechanism for generating new cryptographic key material from some existing, original ("master") key material and potentially other information. Derived keys are used for a variety of purposes including data encryption and message authentication. The reason for doing key derivation itself is typically a combination of a desire to expand a given, but limited, set of original key material and prudent security practices of limiting use (exposure) of such key material. Key separation (such as avoiding use of the same key material for multiple purposes) is an example of such practices.

The key derivation process may be based on passphrases agreed upon or remembered by users, or it can be based on some shared "master" cryptographic keys (and be intended to reduce exposure of such master keys), etc. Derived keys themselves may be used in XML Signature and XML Encryption as any other keys; in particular, they may be used to compute message authentication codes (e.g. digital signatures using symmetric keys) or for encryption/decryption purposes.

5.4.1 ConcatKDF

Identifier:

http://www.w3.org/2009/xmlenc11#ConcatKDF

The ConcatKDF key derivation algorithm, defined in Section 5.8.1 of NIST SP 800-56A [SP800-56A] (and equivalent to the KDF3 function defined in ANSI X9.44-2007 [ANSI-X9-44-2007] when the contents of the `OtherInfo` parameter is structured as in NIST SP 800-56A), takes several parameters. These parameters are represented in the `xenc11:ConcatKDFParamsType`:

```
Schema Definition:

<!-- targetNamespace=&#x27;http://www.w3.org/2009/xmlenc11#&#x27; -->

<!-- use this element type as a child of xenc11:KeyDerivationMethod
     when used with ConcatKDF -->
<element name="ConcatKDFParams" type="xenc11:ConcatKDFParamsType"/>

<complexType name="ConcatKDFParamsType">
  <sequence>
    <element ref="ds:DigestMethod"/>
  </sequence>
  <attribute name="AlgorithmID" type="hexBinary"/>
  <attribute name="PartyUInfo" type="hexBinary"/>
  <attribute name="PartyVInfo" type="hexBinary"/>
  <attribute name="SuppPubInfo" type="hexBinary"/>
  <attribute name="SuppPrivInfo" type="hexBinary"/>
</complexType>
```

The `ds:DigestMethod` element identifies the digest algorithm used by the KDF. Compliant implementations MUST support SHA-256 and SHA-1 (support for SHA-1 is present only for backwards-compatibility reasons). Support for SHA-384 and SHA-512 is OPTIONAL.

Los atributos `AlgorithmID`, `PartyUInfo`, `PartyVInfo`y se definen en [SP800-56A`SuppPubInfo`] . Su presencia es opcional pero DEBE estar PRESENTE para aplicaciones que deben cumplir con [SP800-56A]. Nota: El componente incluirá un nonce cuando ConcatKDF se utilice junto con un esquema de acuerdo de clave estático-estático Diffie-Hellman (o estático-estático ECDH); ver más [SP800-56A]. `SuppPrivInfoAlgorithmIDPartyVInfoPartyUInfo PartyUInfo`

En [SP800-56A], `AlgorithmID`, `PartyUInfo`, y los atributos se definen como cadenas de bits de longitud arbitraria, por lo que `PartyVInfo`es posible que sea necesario rellenarlos para codificarlos en hexBinary para el cifrado XML. SE DEBE utilizar el siguiente método de relleno y codificación al codificar valores de cadena de bits para , , y : `SuppPubInfoSuppPrivInfoAlgorithmIDPartyUInfoPartyVInfoSuppPubInfoSuppPrivInfo`

1. La cadena de bits se divide en octetos utilizando codificación big-endian. Si la longitud de la cadena de bits no es múltiplo de 8, agregue bits de relleno (valor 0) según sea necesario al último octeto para convertirlo en múltiplo de 8.
2. Anteponga un octeto a la cadena de octetos del paso 1. Este octeto identificará (en una representación big-endian) el número de bits de relleno agregados al último octeto en el paso 1.
3. Codifique la cadena de octetos resultante del paso 2 como una cadena hexbinaria.

Ejemplo: la cadena de bits `11011`, que tiene 5 bits de largo, obtiene 3 bits de relleno adicionales para convertirse en la cadena de bits `11011000`(o `D8`en hexadecimal). Luego, a esta cadena de bits se le antepone un octeto que identifica el número de bits de relleno para convertirse en la cadena de octeto (en hexadecimal) `03D8`, que finalmente se codifica como un valor de cadena binaria hexadecimal de "03D8".

Tenga en cuenta que, como se especifica en [SP800-56A], estos atributos se concatenarán para formar una cadena de bits "OtherInfo" que se utiliza con la función de derivación de clave. La concatenación DEBE realizarse utilizando los valores de cadena de bits originales sin relleno". Las aplicaciones también DEBEN verificar que estos atributos, de una manera específica de la aplicación no definida en este documento, identifiquen algoritmos y partes de acuerdo con NIST SP800-56.

A continuación se muestra un ejemplo de un `xenc11:DerivedKey`elemento con este algoritmo de derivación de claves. En este ejemplo, el valor de la cadena de bits `AlgorithmID`es `00000000`, el valor de la cadena de bits `PartyUInfo`es `11011`y el valor de la cadena de bits `PartyVInfo`es `11010`:

EJEMPLO 25

```
<xenc11:DerivedKey xmlns:xsi = "http://www.w3.org/2001/XMLSchema-instance" xmlns:ds = "http://www.w3.org/2000/09/xmldsig#" x
```

NOTA
Si bien se puede usar cualquier cadena de bits con ConcatKDF, se RECOMIENDA mantener los bytes alineados para una mayor interoperabilidad.

5.4.2 PBKDF2

Identificador:
http://www.w3.org/2009/xmlenc11#pbkdf2

El algoritmo de derivación de claves PBKDF2 y las definiciones de tipo ASN.1 para sus parámetros se definen en PKCS #5 v2.0 [PKCS5]. Las definiciones del esquema XML para los parámetros se definen en [PKCS5Amd1] y las mismas se pueden especificar encerrándolas dentro de un `xenc11:PBKDF2-params`elemento secundario del `xenc11:KeyDerivationMethod`elemento.

```
Definición del esquema :

< nombre del elemento = "PBKDF2-params" tipo = "xenc11:PBKDF2ParameterType" />

<complexType nombre = "PBKDF2ParameterType" > <secuencia> <elemento nombre = "Salt" > <complexType> <elección> <elemento nombr

<complexType nombre = "AlgorithmIdentifierType" > <secuencia> <elemento nombre = "Parámetros" tipo = "anyType" minOccurs = "0"

<complexType nombre = "PRFAlgorithmIdentifierType" > <complexContent> <restricción base = "xenc11:AlgorithmIdentifierType" > <
```

(Nota: se agregó una nueva línea al atributo Algoritmo para que quepa en esta página, pero no forma parte del URI).

The `PBKDF2-params` element and its child elements have the same names and meaning as the corresponding components of the `PBKDF2-params` ASN.1 type in [PKCS5]. Note, in case of ConcatKDF and the Diffie Hellman legacy KDF, `KeyLength` is an implied parameter and needs to be inferred from the context, but in the case of PBKDF2 the `KeyLength` child element has to be specified, as it has been made a mandatory parameter to be consistent with PKCS5. For PBKDF2, the inferred key length must match the specified key length, otherwise it is an error condition.

The `AlgorithmIdentifierType` corresponds to the `AlgorithmIdentifier` type of [PKCS5] and carries the algorithm identifier in the `Algorithm` attribute. Algorithm specific parameters, where applicable, can be specified using the `Parameters` element.

The `PRFAlgorithmIdentifierType` is derived from the `AlgorithmIdentifierType` and constrains the choice of algorithms to those contained in the PBKDF2-PRFs set defined in [PKCS5]. This type is used to specify a pseudorandom function (PRF) for PBKDF2. Whereas HMAC-SHA1 is the default PRF algorithm in [PKCS5], use of HMAC-SHA256 is RECOMMENDED by this specification (see [XMLDSIG-CORE1], [HMAC]).

An example of an `xenc11:DerivedKey` element with this key derivation algorithm is:

EXAMPLE 26

```
<xenc11:DerivedKey
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:xenc="http://www.w3.org/2001/04/xmlenc#"
    xmlns:xenc11="http://www.w3.org/2009/xmlenc11#">
  <xenc11:KeyDerivationMethod Algorithm="http://www.w3.org/2009/xmlenc11#pbkdf2"/>
  <xenc11:PBKDF2-params>
    <xenc11:Salt>
      <xenc11:Specified>Df3dRAhjGh8=</xenc11:Specified>
    </xenc11:Salt>
    <xenc11:IterationCount>2000</xenc11:IterationCount>
    <xenc11:KeyLength>16</xenc11:KeyLength>
    <xenc11:PRF Algorithm="http://www.w3.org/2001/04/xmldsig-more#hmac-sha256"/>
  </xenc11:PBKDF2-params>
</xenc11:KeyDerivationMethod>
<xenc:ReferenceList>
  <xenc:DataReference URI="#ED"/>
</xenc:ReferenceList>
<xenc11:MasterKeyName>Our shared secret</xenc11:MasterKeyName>
</xenc11:DerivedKey>
```

5.5 Key Transport

Key Transport algorithms are public key encryption algorithms especially specified for encrypting and decrypting keys. Their identifiers appear as `Algorithm` attributes to `EncryptionMethod` elements that are children of `EncryptedKey`. `EncryptedKey` is in turn the child of a `ds:KeyInfo` element. The type of key being transported, that is to say the algorithm in which it is planned to use the transported key, is given by the `Algorithm` attribute of the `EncryptionMethod` child of the `EncryptedData` or `EncryptedKey` parent of this `ds:KeyInfo` element.

(Key Transport algorithms may optionally be used to encrypt data in which case they appear directly as the `Algorithm` attribute of an `EncryptionMethod` child of an `EncryptedData` element. Because they use public key algorithms directly, Key Transport algorithms are not efficient for the transport of any amounts of data significantly larger than symmetric keys.)

5.5.1 RSA Version 1.5

Identifier:
  http://www.w3.org/2001/04/xmlenc#rsa-1_5

The RSAES-PKCS1-v1_5 algorithm, specified in RFC 3447 [PKCS1], takes no explicit parameters. An example of an RSA Version 1.5 `EncryptionMethod` element is:

EXAMPLE 27

```
<EncryptionMethod Algorithm="http://www.w3.org/2001/04/xmlenc#rsa-1_5" />
```

The `CipherValue` for such an encrypted key is the base64 [RFC2045] encoding of the octet string computed as per RFC 3447 [PKCS1], section 7.2.1: Encryption operation]. As specified in the EME-PKCS1-v1_5 function RFC 3447 [PKCS1], section 7.2.1, the value input to the key transport function is as follows:

EXAMPLE 28

```
CRYPT ( PAD ( KEY ))
```

where the padding is of the following special form:

EXAMPLE 29

```
02 | PS* | 00 | key
```

where "|" is concatenation, "02" and "00" are fixed octets of the corresponding hexadecimal value, PS is a string of strong pseudo-random octets [RANDOM] at least eight octets long, containing no zero octets, and long enough that the value of the quantity being CRYPTed is one octet shorter than the RSA modulus, and "key" is the key being transported. The key is 192 bits for TRIPLEDES and 128, 192, or 256 bits for AES.

Implementations MUST support this key transport algorithm for transporting 192-bit TRIPLEDES keys. Support of this algorithm for transporting other keys is OPTIONAL. RSA-OAEP is RECOMMENDED for the transport of AES keys.

The resulting base64 [RFC2045] string is the value of the child text node of the `CipherData` element, e.g.

EXAMPLE 30

```
<CipherData>
  <CipherValue>IWijxQjUrcXBYoCei4QxjWo9Kg8D3p9tlWoT4
t0/gyTE96639In0FZFY2/rvP+/bMJ01EArmKZsR5VW3rwoPxw=</CipherValue>
</CipherData>
```

(Note: A newline has been added to the `CipherValue` to fit on this page, but is not part of value.)

Note: Implementation of RSA v1.5 is NOT RECOMMENDED due to security risks associated with the algorithm.

5.5.2 RSA-OAEP

Identifier:
  http://www.w3.org/2001/04/xmlenc#rsa-oaep-mgf1p (including `MGF1 with SHA1` mask generation function)
Identifier:
  http://www.w3.org/2009/xmlenc11#rsa-oaep

El algoritmo RSAES-OAEP-ENCRYPT, como se especifica en RFC 3447 [PKCS1], tiene opciones que definen la función de resumen del mensaje y la función de generación de máscara, así como un `PSourceAlgorithm` parámetro opcional. Los valores predeterminados definidos en RFC 3447 son `SHA1` para el resumen del mensaje y `MGF1 with SHA1`para la función de generación de máscara. Tanto las funciones de resumen de mensajes como de generación de máscaras se utilizan en la operación EME-OAEP-ENCODE como parte de RSAES-OAEP-ENCRYPT.

El identificador http://www.w3.org/2001/04/xmlenc#rsa-oaep-mgf1p define la función de generación de máscara como el valor fijo de `MGF1 with SHA1`. En este caso NO DEBE proporcionarse el `xenc11:MGF`elemento opcional del `xenc:EncryptionMethod`elemento .

El identificador http://www.w3.org/2009/xmlenc11#rsa-oaep define la función de generación de máscara utilizando el `xenc11:MGF` elemento opcional del `xenc:EncryptionMethod`elemento. Si no está presente, `MGF1 with SHA1`se utilizará el valor predeterminado de.

Los siguientes URI definen los distintos valores de URI de la función de generación de máscaras que se pueden utilizar. Estos corresponden a los identificadores de objetos definidos en RFC 4055 [RFC4055]:

- MGF1 con SHA1: http://www.w3.org/2009/xmlenc11#mgf1sha1
- MGF1 con SHA224: http://www.w3.org/2009/xmlenc11#mgf1sha224
- MGF1 con SHA256: http://www.w3.org/2009/xmlenc11#mgf1sha256

- MGF1 con SHA384: http://www.w3.org/2009/xmlenc11#mgf1sha384
- MGF1 con SHA512: http://www.w3.org/2009/xmlenc11#mgf1sha512

De lo contrario, los dos identificadores definen el mismo uso del algoritmo RSA-OAEP, como sigue.

La función de resumen del mensaje DEBE especificarse utilizando el atributo Algoritmo del `ds:DigestMethod`elemento secundario del `xenc:EncryptionMethod`elemento. Si no se especifica, `SHA1`se utilizará el valor predeterminado de.

`PSourceAlgorithm`El valor del parámetro RSA-OAEP opcional PUEDE proporcionarse explícitamente colocando los octetos codificados en base64 en el `xenc:OAEPparams`elemento XML.

La definición y descripción del esquema XML Encryption 1.0 para el `EncryptionMethod`elemento se encuentran en la sección 3.2 El elemento EncryptionMethod . A continuación se muestra la adición de XML Encryption 1.1 para el tipo MGF:

```
Definición del esquema :

< nombre del elemento = "MGF" tipo = "xenc11:MGFType" />

<complexType nombre = "MGFType" > <complexContent> <restricción base = "xenc11:AlgorithmIdentifierType" > <atributo nombre = "
```

Un ejemplo de un elemento RSA-OAEP es:

EJEMPLO 31

```
< Algoritmo de método de cifrado = "http://www.w3.org/2001/04/xmlenc#rsa-oaep-mgf1p" > <OAEPparams> 9lWu3Q== </OAEPparams> <
```

EJEMPLO 32

```
< Algoritmo de método de cifrado = "http://www.w3.org/2001/04/xmlenc#rsa-oaep-mgf1p" > <OAEPparams> 9lWu3Q== </OAEPparams> <
```

Otro ejemplo es:

EJEMPLO 33

```
< Algoritmo de método de cifrado = "http://www.w3.org/2009/xmlenc11#rsa-oaep" > <OAEPparams> 9lWu3Q== </OAEPparams> <xenc11:
```

La `CipherValue`clave cifrada RSA-OAEP es la codificación base64 [RFC2045] de la cadena de octetos calculada según RFC 3447 [PKCS1], sección 7.1.1: Operación de cifrado. Como se describe en la función EME-OAEP-ENCODE RFC 3447 [PKCS1], sección 7.1.1, el valor ingresado a la función de transporte de claves se calcula usando la función de resumen de mensajes y la cadena especificada en los elementos `DigestMethod`y `OAEPparams` y usando la función de generador de máscaras. especificado con el `xenc11:MGF`elemento o el valor predeterminado `MGF1 with SHA1`especificado en RFC 3447. La longitud de salida deseada para EME-OAEP-ENCODE es un byte más corto que el módulo RSA.

El tamaño de la clave transportada es de 192 bits para TRIPLEDES y de 128, 192 o 256 bits para AES. Las implementaciones DEBEN implementar RSA-OAEP para el transporte de todos los tipos y tamaños de claves que son obligatorios para el cifrado simétrico. PUEDEN IMPLEMENTAR RSA-OAEP para el transporte de otras claves.

5.6 Acuerdo clave

A Key Agreement algorithm provides for the derivation of a shared secret key based on a shared secret computed from certain types of compatible public keys from both the sender and the recipient. Information from the originator to determine the secret is indicated by an optional `OriginatorKeyInfo` parameter child of an `AgreementMethod` element while that associated with the recipient is indicated by an optional `RecipientKeyInfo`. A shared key is derived from this shared secret by a method determined by the Key Agreement algorithm.

Note: XML Encryption does not provide an online key agreement negotiation protocol. The `AgreementMethod` element can be used by the originator to identify the keys and computational procedure that were used to obtain a shared encryption key. The method used to obtain or select the keys or algorithm used for the agreement computation is beyond the scope of this specification.

The `AgreementMethod` element appears as the content of a `ds:KeyInfo` since, like other `ds:KeyInfo` children, it yields a key. This `ds:KeyInfo` is in turn a child of an `EncryptedData` or `EncryptedKey` element. The `Algorithm` attribute and `KeySize` child of the `EncryptionMethod` element under this `EncryptedData` or `EncryptedKey` element are implicit parameters to the key agreement computation. In cases where this `EncryptionMethod` algorithm URI is insufficient to determine the key length, a `KeySize` MUST have been included.

Key derivation algorithms (with associated parameters) may be explicitly declared by using the `xenc11:KeyDerivationMethod` element. This element will then be placed at the extensibility point of the `xenc:AgreementMethodType` (see below).

In addition, the sender may place a `KA-Nonce` element under `AgreementMethod` to assure that different keying material is generated even for repeated agreements using the same sender and recipient public keys. For example:

EXAMPLE 34

```
<EncryptedData>
  <EncryptionMethod Algorithm="Example:Block/Alg">
```

```
      <KeySize>80</KeySize>
    </EncryptionMethod>
    <ds:KeyInfo xmlns:ds="http://www.w3.org/2000/09/xmldsig#">
      <AgreementMethod Algorithm="example:Agreement/Algorithm">
        <KA-Nonce>Zm9v</KA-Nonce>
        <xenc11:KeyDerivationMethod
            Algorithm="http://www.w3.org/2009/xmlenc11#ConcatKDF">
          <xenc11:ConcatKDFParams
              AlgorithmID="00" PartyUInfo="" PartyVInfo="">
            <ds:DigestMethod
                Algorithm="http://www.w3.org/2001/04/xmlenc#sha256"/>
          </xenc11:ConcatKDFParams>
        </xenc11:KeyDerivationMethod>

        <OriginatorKeyInfo>
          <ds:KeyValue>....</ds:KeyValue>
        </OriginatorKeyInfo>
        <RecipientKeyInfo>
          <ds:KeyValue>....</ds:KeyValue>
        </RecipientKeyInfo>
      </AgreementMethod>
    </ds:KeyInfo>
    <CipherData>...</CipherData>
  </EncryptedData>
```

If the agreed key is being used to wrap a key, rather than data as above, then `AgreementMethod` would appear inside a `ds:KeyInfo` inside an `EncryptedKey` element.

The Schema for `AgreementMethod` is as follows:

Schema Definition:

```
<element name="AgreementMethod" type="xenc:AgreementMethodType"/>

<complexType name="AgreementMethodType" mixed="true">
  <sequence>
    <element name="KA-Nonce" minOccurs="0" type="base64Binary"/>
    <!-- <element ref="ds:DigestMethod" minOccurs="0"/> -->
    <any namespace="##other" minOccurs="0" maxOccurs="unbounded"/>
    <element name="OriginatorKeyInfo" minOccurs="0"
             type="ds:KeyInfoType"/>
    <element name="RecipientKeyInfo" minOccurs="0"
             type="ds:KeyInfoType"/>
  </sequence>
  <attribute name="Algorithm" type="anyURI" use="required"/>
</complexType>
```

5.6.1 Diffie-Hellman Key Values

Identifier:
http://www.w3.org/2001/04/xmlenc#DHKeyValue

Diffie-Hellman keys can appear directly within `KeyValue` elements or be obtained by `ds:RetrievalMethod` fetches as well as appearing in certificates and the like. The above identifier can be used as the value of the `Type` attribute of `Reference` or `ds:RetrievalMethod` elements.

As specified in [ESDH], a DH public key consists of up to six quantities, two large primes p and q, a "generator" g, the public key, and validation parameters "seed" and "pgenCounter". These relate as follows: The public key = (g**x mod p) where x is the corresponding private key; p = j*q + 1 where j >= 2. "seed" and "pgenCounter" are optional and can be used to determine if the Diffie-Hellman key has been generated in conformance with the algorithm specified in [ESDH]. Because the primes and generator can be safely shared over many DH keys, they may be known from the application environment and are optional. The schema for a `DHKeyValue` is as follows:

Schema Definition:

```
<element name="DHKeyValue" type="xenc:DHKeyValueType"/>

<complexType name="DHKeyValueType">
  <sequence>
    <sequence minOccurs="0">
      <element name="P" type="ds:CryptoBinary"/>
      <element name="Q" type="ds:CryptoBinary"/>
      <element name="Generator"type="ds:CryptoBinary"/>
    </sequence>
    <element name="Public" type="ds:CryptoBinary"/>
    <sequence minOccurs="0">
      <element name="seed" type="ds:CryptoBinary"/>
      <element name="pgenCounter" type="ds:CryptoBinary"/>
    </sequence>
  </sequence>
</complexType>
```

5.6.2 Diffie-Hellman Key Agreement

The Diffie-Hellman (DH) key agreement protocol [ESDH] involves the derivation of shared secret information based on compatible DH keys from the sender and recipient. Two DH public keys are compatible if they have the same prime and generator. If, for the second one, `Y = g**y mod p`, then the two parties can calculate the shared secret `ZZ = ( g**(x*y) mod p )` even though each knows only their own private key and the other party's public key. Leading zero bytes MUST be maintained in `ZZ` so it will be the same length, in bytes, as `p`. The size of `p` MUST be at least 512 bits and `g` at least 160 bits. There are numerous other complex security considerations in the selection of `g`, `p`, and a random `x` as described in [ESDH].

The Diffie-Hellman shared secret `zz` is used as the input to a KDF to produce a secret key. XML Signature 1.0 defined a specific KDF to be used with Diffie-Hellman; that KDF is now known as the "Legacy KDF" and is defined in Section 5.6.2.2. Use of Diffie-Hellman with explicit KDFs is described in Section 5.6.2.1.

Implementation of Diffie-Hellman key agreement is OPTIONAL. However, if implemented, such implementations MUST support the Legacy Key Derivation Function and SHOULD support Diffie-Hellman with explicit Key Derivation Functions

An example of a DH `AgreementMethod` element using the Legacy Key Derivation Function (Section 5.6.2.2) is as follows:

EXAMPLE 35

```
<AgreementMethod
    Algorithm="http://www.w3.org/2001/04/xmlenc#dh"
    ds:xmlns="http://www.w3.org/2000/09/xmldsig#">
  <KA-Nonce>Zm9v</KA-Nonce>
  <ds:DigestMethod Algorithm="http://www.w3.org/2000/09/xmldsig#sha1"/>
  <OriginatorKeyInfo>
    <ds:X509Data>
      <ds:X509Certificate>...</ds:X509Certificate>
    </ds:X509Data>
  </OriginatorKeyInfo>
  <RecipientKeyInfo>
    <ds:KeyValue>...</ds:KeyValue>
  </RecipientKeyInfo>
</AgreementMethod>
```

5.6.2.1 Diffie-Hellman Key Agreement with Explicit Key Derivation Functions

Identifier:
: http://www.w3.org/2009/xmlenc11#dh-es

It is RECOMMENDED that the shared key material for a Diffie-Hellman key agreement be calculated from the Diffie-Hellman shared secret using a key derivation function (KDF) in accordance with Section 5.4.

An example of a DH `AgreementMethod` element using an explicit key derivation function is as follows:

EXAMPLE 36

```
<xenc:AgreementMethod Algorithm="http://www.w3.org/2009/xmlenc11#dh-es">
  <xenc11:KeyDerivationMethod Algorithm="http://www.w3.org/2009/xmlenc11#ConcatKDF">
    <xenc11:ConcatKDFParams AlgorithmID="00" PartyUInfo="" PartyVInfo="">
      <ds:DigestMethod Algorithm="http://www.w3.org/2001/04/xmlenc#sha256"/>
    </xenc11:ConcatKDFParams>
  </xenc11:KeyDerivationMethod>
  <xenc:OriginatorKeyInfo>
    <ds:X509Data>
      <ds:X509Certificate><!-- X.509 Certificate here --></ds:X509Certificate>
    </ds:X509Data>
  </xenc:OriginatorKeyInfo>
  <xenc:RecipientKeyInfo>
    <ds:X509Data>
      <ds:X509SKI></ds:X509SKI>
      <!-- hint for the recipient's private key -->
    </ds:X509Data>
  </xenc:RecipientKeyInfo>
</xenc:AgreementMethod>
```

5.6.2.2 Diffie-Hellman Key Agreement with Legacy Key Derivation Function

Identifier:
: http://www.w3.org/2001/04/xmlenc#dh

XML Signature 1.0 defined a specific KDF for use with Diffie-Hellman key agreement. In order to guarantee interoperability, implementations that choose to implement Diffie-Hellman MUST support the use of the Diffie-Hellman Legacy KDF defined in this section.

Assume that the Diffie-Hellman shared secret is the octet sequence `ZZ`. The Diffie-Hellman Legacy KDF calculates the shared keying material as follows:

EXAMPLE 37

```
Keying Material = KM(1) | KM(2) | ...
```

where "|" is byte stream concatenation and

EXAMPLE 38

```
KM(counter) = DigestAlg ( ZZ | counter | EncryptionAlg |
            KA-Nonce | KeySize )
```

`DigestAlg`
: The message digest algorithm specified by the `DigestMethod` child of `AgreementMethod`.

`EncryptionAlg`
: The URI of the encryption algorithm, including possible key wrap algorithms, in which the derived keying material is to be used ("Example:Block/Alg" in the example above), not the URI of the agreement algorithm. This is the value of the `Algorithm` attribute of the `EncryptionMethod` child of the `EncryptedData` or `EncryptedKey` grandparent of `AgreementMethod`.

`KA-Nonce`
: The base64 decoding the content of the `KA-Nonce` child of `AgreementMethod`, if present. If the `KA-Nonce` element is absent, it is null.

`Counter`

A one byte counter starting at one and incrementing by one. It is expressed as two hex digits where letters A through F are in upper case.

`KeySize`
: The size in bits of the key to be derived from the shared secret as the UTF-8 string for the corresponding decimal integer with only digits in the string and no leading zeros. For some algorithms the key size is inherent in the URI. For others, such as most stream ciphers, it must be explicitly provided.

For example, the initial `(KM(1))` calculation for the `EncryptionMethod` of the Key Agreement example (section 5.5) would be as follows, where the binary one byte counter value of 1 is represented by the two character UTF-8 sequence `01`, `ZZ` is the shared secret, and "`foo`" is the base64 decoding of "`Zm9v`".

EXAMPLE 39

```
SHA-1 ( ZZ01Example:Block/Algfoo80 )
```

Assuming that `ZZ` is `0xDEADBEEF`, that would be

EXAMPLE 40

```
SHA-1( 0xDEADBEEF30314578616D706C653A426C6F636B2F416C67666F6F3830 )
```

whose value is

EXAMPLE 41

```
0x534C9B8C4ABDCB50038B42015A181711068B08C1
```

Each application of `DigestAlg` for successive values of `Counter` will produce some additional number of bytes of keying material. From the concatenated string of one or more `KM`'s, enough leading bytes are taken to meet the need for an actual key and the remainder discarded. For example, if `DigestAlg` is SHA-1 which produces 20 octets of hash, then for 128 bit AES the first 16 bytes from `KM(1)` would be taken and the remaining 4 bytes discarded. For 256 bit AES, all of `KM(1)` suffixed with the first 12 bytes of KM(2) would be taken and the remaining 8 bytes of `KM(2)` discarded.

5.6.3 Elliptic Curve Diffie-Hellman (ECDH) Key Values

Identifier:
: http://www.w3.org/2009/xmldsig11#ECKeyValue

ECDH has identical public key parameters as ECDSA and can be represented with the `ECKeyValue` element [XMLDSIG-CORE1]. Note that if the curve parameters are explicitly stated using the ECParameters element, then the Cofactor element MUST be included.

As with Diffie-Hellman keys, Elliptic Curve Key Values can appear directly within `KeyValue` elements or be obtained by `ds:RetrievalMethod` fetches as well as appearing in certificates and the like. The above identifier can be used as the value of the `Type` attribute of `Reference` or `ds:RetrievalMethod` elements.

5.6.4 Elliptic Curve Diffie-Hellman (ECDH) Key Agreement (Ephemeral-Static Mode)

Identifier:
: http://www.w3.org/2009/xmlenc11#ECDH-ES

ECDH is the elliptic curve analogue to the Diffie-Hellman key agreement algorithm. Details of the ECDH primitive can be found in [ECC-ALGS]. When ECDH is used in Ephemeral-Static (ES) mode, the recipient has a static key pair, but the sender generates a ephemeral key pair for each message. The same ephemeral key may be used when there are multiple recipients that use the same curve parameters.

Compliant implementations are REQUIRED to support ECDH-ES key agreement using the P-256 prime curve specified in Section D.2.3 of FIPS 186-3 [FIPS-186-3]. (This is the same curve that is REQUIRED in XML Signature 1.1 to be supported for the ECDSAwithSHA256 algorithm.) It is further RECOMMENDED that implementations also support the P-384 and P-521 prime curves for ECDH-ES; these curves are defined in Sections D.2.4 and D.2.5 of FIPS 186-3, respectively.

The shared key material is calculated from the Diffie-Hellman shared secret using a key derivation function (KDF). While applications may define other KDFs, compliant implementations MUST implement ConcatKDF (see section 5.4.1 ConcatKDF). An example of `xenc:EncryptedData` using the ECDH-ES key agreement algorithm with the ConcatKDF key derivation algorithm is as follows:

EXAMPLE 42

```
<xenc:EncryptedData
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:xenc="http://www.w3.org/2001/04/xmlenc#"
    xmlns:ds="http://www.w3.org/2000/09/xmldsig#"
    xmlns:dsig11="http://www.w3.org/2009/xmldsig11#"
    xmlns:xenc11="http://www.w3.org/2009/xmlenc11#"
    Type="http://www.w3.org/2001/04/xmlenc#">

  <xenc:EncryptionMethod Algorithm="http://www.w3.org/2001/04/xmlenc#aes128-cbc" />
  <!-- describes the encrypted AES content encryption key -->
  <ds:KeyInfo>
    <xenc:EncryptedKey>
      <xenc:EncryptionMethod Algorithm="http://www.w3.org/2001/04/xmlenc#kw-aes128"/>
      <!-- describes the key encryption key -->
      <ds:KeyInfo>
        <xenc:AgreementMethod Algorithm="http://www.w3.org/2009/xmlenc11#ECDH-ES">
          <xenc11:KeyDerivationMethod Algorithm="http://www.w3.org/2009/xmlenc11#ConcatKDF">
            <xenc11:ConcatKDFParams AlgorithmID="00" PartyUInfo="" PartyVInfo="">
              <ds:DigestMethod Algorithm="http://www.w3.org/2001/04/xmlenc#sha256"/>
            </xenc11:ConcatKDFParams>
```

```
                </xenc11:KeyDerivationMethod>
                <xenc:OriginatorKeyInfo>
                  <ds:KeyValue>
                    <dsig11:ECKeyValue>
                      <!-- ephemeral ECC public key of the originator -->
                    </dsig11:ECKeyValue>
                  </ds:KeyValue>
                </xenc:OriginatorKeyInfo>
                <xenc:RecipientKeyInfo>
                  <ds:X509Data>
                    <ds:X509SKI></ds:X509SKI>
                    <!-- hint for the recipient's private key -->
                  </ds:X509Data>
                </xenc:RecipientKeyInfo>
              </xenc:AgreementMethod>
            </ds:KeyInfo>
            <xenc:CipherData>
              <xenc:CipherValue><!-- encrypted AES content encryption key --></xenc:CipherValue>
            </xenc:CipherData>
          </xenc:EncryptedKey>
        </ds:KeyInfo>

        <xenc:CipherData>
          <xenc:CipherValue>
            <!-- encrypted data -->
          </xenc:CipherValue>
        </xenc:CipherData>

      </xenc:EncryptedData>
```

5.7 Symmetric Key Wrap

Symmetric Key Wrap algorithms are shared secret key encryption algorithms especially specified for encrypting and decrypting symmetric keys. When wrapped keys are used, then an `EncryptedKey` element will appear as a child of a `ds:KeyInfo` element. This `EncryptedKey` element will have an `EncryptionMethod` child whose `Algorithm` attribute in turn identifies the key wrap algorithm.

The algorithm for which the encrypted key is intended depends on the context of the `ds:KeyInfo` element: `ds:KeyInfo` can occur as a child of either an `EncryptedData` or `EncryptedKey` element; in both cases, `ds:KeyInfo` will have an `EncryptionMethod` sibling that identifies the algorithm.

EXAMPLE 43

```
<EncryptedData |EncryptedKey>
  <EncryptionMethod Algorithm="@alg1"/>
  <ds:KeyInfo>
    <EncryptedKey>
      <EncryptionMethod Algorithm="@alg2"/>
    </EncryptedKey>
  </ds:KeyInfo>
</EncryptedData |EncryptedKey>
```

5.7.1 CMS Triple DES Key Wrap

Identifiers:
http://www.w3.org/2001/04/xmlenc#kw-tripledes

XML Encryption implementations MUST support TRIPLEDES wrapping of 168 bit keys as described in [CMS-WRAP] and may optionally support TRIPLEDES wrapping of other keys.

An example of a TRIPLEDES Key Wrap `EncryptionMethod` element is as follows:

EXAMPLE 44

```
<EncryptionMethod Algorithm="http://www.w3.org/2001/04/xmlenc#kw-tripledes" />
```

5.7.2 AES KeyWrap

Identifiers:
http://www.w3.org/2001/04/xmlenc#kw-aes128
http://www.w3.org/2001/04/xmlenc#kw-aes192
http://www.w3.org/2001/04/xmlenc#kw-aes256

Implementation of AES key wrap is described in [AES-WRAP]. It provides for confidentiality and integrity. This algorithm is defined only for inputs which are a multiple of 64 bits. The information wrapped need not actually be a key. The algorithm is the same whatever the size of the AES key used in wrapping, called the key encrypting key or KEK. The implementation requirements are indicated below.

128 bit AES Key Encrypting Key
Implementation of wrapping 128 bit keys REQUIRED.
Wrapping of other key sizes OPTIONAL.
192 bit AES Key Encrypting Key
All support OPTIONAL.
256 bit AES Key Encrypting Key
Implementation of wrapping 256 bit keys REQUIRED.
Wrapping of other key sizes OPTIONAL.

5.8 Message Digest

Message digest algorithms can be used in `AgreementMethod` as part of the key derivation, within RSA-OAEP encryption as a hash function, and in connection with the HMAC message authentication code method [HMAC] as described in [XMLDSIG-CORE1].) Use of SHA-256 is strongly recommended over SHA-1 because recent advances in cryptanalysis (see e.g. [SHA-1-Analysis], [SHA-1-Collisions]) have cast doubt on the long-term collision resistance of SHA-1. Therefore, SHA-1 support is REQUIRED in this specification only for backwards-compatibility reasons.

5.8.1 SHA1

Identifier:
: http://www.w3.org/2000/09/xmldsig#sha1

The SHA-1 algorithm [FIPS-180-3] takes no explicit parameters. An example of an SHA-1 `DigestMethod` element is:

EXAMPLE 45

```
<DigestMethod Algorithm="http://www.w3.org/2000/09/xmldsig#sha1" />
```

A SHA-1 digest is a 160-bit string. The content of the `DigestValue` element shall be the base64 encoding of this bit string viewed as a 20-octet octet stream. For example, the `DigestValue` element for the message digest:

EXAMPLE 46

```
A9993E36 4706816A BA3E2571 7850C26C 9CD0D89D
```

from Appendix A of the SHA-1 standard would be:

EXAMPLE 47

```
<DigestValue>qZk+NkcGgWq6PiVxeFDCbJzQ2J0=</DigestValue>
```

5.8.2 SHA256

Identifier:
: http://www.w3.org/2001/04/xmlenc#sha256

The SHA-256 algorithm [FIPS-180-3] takes no explicit parameters. An example of an SHA-256 `DigestMethod` element is:

EXAMPLE 48

```
<DigestMethod Algorithm="http://www.w3.org/2001/04/xmlenc#sha256" />
```

A SHA-256 digest is a 256-bit string. The content of the `DigestValue` element shall be the base64 encoding of this bit string viewed as a 32-octet octet stream.

5.8.3 SHA384

Identifier:
: http://www.w3.org/2001/04/xmlenc#sha384

The SHA-384 algorithm [FIPS-180-3] takes no explicit parameters. An example of an SHA-384 `DigestMethod` element is:

EXAMPLE 49

```
<DigestMethod Algorithm="http://www.w3.org/2001/04/xmlenc#sha384" />
```

A SHA-384 digest is a 384-bit string. The content of the `DigestValue` element shall be the base64 encoding of this bit string viewed as a 48-octet octet stream.

5.8.4 SHA512

Identifier:
: http://www.w3.org/2001/04/xmlenc#sha512

The SHA-512 algorithm [FIPS-180-3] takes no explicit parameters. An example of an SHA-512 `DigestMethod` element is:

EXAMPLE 50

```
<DigestMethod Algorithm="http://www.w3.org/2001/04/xmlenc#sha512" />
```

A SHA-512 digest is a 512-bit string. The content of the `DigestValue` element shall be the base64 encoding of this bit string viewed as a 64-octet octet stream.

5.8.5 RIPEMD-160

Identifier:
: http://www.w3.org/2001/04/xmlenc#ripemd160

The RIPEMD-160 algorithm [RIPEMD-160] takes no explicit parameters. An example of an RIPEMD-160 `DigestMethod` element is:

EXAMPLE 51

```
<DigestMethod Algorithm="http://www.w3.org/2001/04/xmlenc#ripemd160" />
```

A RIPEMD-160 digest is a 160-bit string. The content of the `DigestValue` element shall be the base64 encoding of this bit string viewed as a 20-octet octet stream.

5.9 Canonicalization

A Canonicalization of XML is a method of consistently serializing XML into an octet stream as is necessary prior to encrypting XML.

5.9.1 Inclusive Canonicalization

Identifiers:
http://www.w3.org/TR/2001/REC-xml-c14n-20010315
http://www.w3.org/TR/2001/REC-xml-c14n-20010315#WithComments
http://www.w3.org/2006/12/xml-c14n11
http://www.w3.org/2006/12/xml-c14n11#WithComments

Canonical XML [XML-C14N11] is a method of serializing XML which includes the in scope namespace and xml namespace attribute context from ancestors of the XML being serialized.

If XML is to be encrypted and then later decrypted into a different environment and it is desired to preserve namespace prefix bindings and the value of attributes in the "xml" namespace of its original environment, then the canonical XML with comments version of the XML should be the serialization that is encrypted.

5.9.2 Exclusive Canonicalization

Identifiers:
http://www.w3.org/2001/10/xml-exc-c14n#
http://www.w3.org/2001/10/xml-exc-c14n#WithComments

Exclusive XML Canonicalization [XML-EXC-C14N] serializes XML in such a way as to include to the minimum extent practical the namespace prefix binding and xml namespace attribute context inherited from ancestor elements.

It is the recommended method where the outer context of a fragment which was signed and then encrypted may be changed. Otherwise the validation of the signature over the fragment may fail because the canonicalization by signature validation may include unnecessary namespaces into the fragment.

6. Security Considerations

6.1 Chosen-Ciphertext Attacks

A number of chosen-ciphertext attacks against implementations of this specification have been published and demonstrated. They all involve the following elements:

1. The attacker knows about the format of the cleartext.
2. The attacker is able to submit substantial numbers of ciphertext messages.
3. The attacker is able to send arbitrary ciphertext, based on previous results.
4. The attacker is able to force the server to use the same key (secret key by CBC-based attacks and server's private key by PKCS#1.5 attacks) for processing of the adapted ciphertext.
5. The server attempting to decrypt the ciphertext in some way signals whether the decrypted text is well-formed or not.

The attacker uses the knowledge of the format and the information about well-formedness to construct a series of ciphertext guesses which reveal the plaintext with much less work than brute force. Attacks of this type have been demonstrated against symmetric encryption using CBC mode [XMLENC-CBC-ATTACK][XMLENC-CBC-ATTACK-COUNTERMEASURES] and on PKCS#1 v1.5. Other future attacks can be expected whenever these conditions are met.

6.1.1 Attacks against the encrypted data (`<EncryptedData>` part)

Using the CBC-based chosen-ciphertext attacks, the attacker sends to the server an XML document with modified encrypted data in the symmetric part (`<EncryptedData>`). After a few requests, the attacker is able to get the whole cleartext without knowledge of the symmetric key.

It would seem that these attacks can be countered by by disrupting any of the conditions, however in practice only preventing condition 3 (sending arbitrary ciphertext) is fully effective. To counter condition 3, it is necessary for the decrypting system to require authenticated integrity protection over the ciphertext. However, unless the mechanism used is bound to the encryption key, there will no way to be sure that the signer is not attempting to recover the plaintext. The simplest and most efficient way to do this is to use an authenticating block mode, such as GCM. An alternative would be an HMAC based on the encryption key over the ciphertext, but it is less efficient and provides no advantages.

Other countermeasures are not likely to be effective. Limiting the number of messages presented or the number of messages using the same key is not practical in large server farms. Attackers can spread their attempts over different servers and long or short periods of time, to foil attempts to detect attacks in progress or determine the location of the attacker.

Signaling well-formedness can occur by emitting different messages for distinct security errors or by exhibiting timing differences. Implementations should avoid these practices, however that is not sufficient to prevent such attacks in an XML protocol environment, such as SOAP. Using a technique called encryption wrapping, the attacker can insert the ciphertext in some schema-legal part of the message. If

the decryption code notices a format error, an error will be returned, but if not the message will be passed to the application which will ignore the bogus plaintext and ultimately respond with an application level success or failure message.

6.1.2 Attacks against the encrypted key (Bleichenbacher's Million question attack on PKCS#1.5)

The goal of the attacker applying the Bleichenbacher's attack is to get the symmetric secret key, which is encrypted in the `<EncryptedKey>` part. Afterward, he would be able to decrypt the whole data carried in the `<EncryptedData>` part.

The basic idea of this attack is to modify the data in the `<EncryptedKey>` part, send the document to the server, and observe if the modified ciphertext contains PKCS#1.5 conformant data. This can be done by:

1. Observing fault messages of the server notifying directly that the request was not PKCS#1.5 conformant (this should not happen).
2. Enlarging the data in the `<EncryptedData>` part and observing the timing differences between inclusion of PKCS-valid and PKCS-invalid keys: if the key is PKCS-valid, the session key is extracted, and the large data is decrypted. Otherwise, the session key cannot be extracted and the large data is not processed, which yields a timing difference.
3. Making specific modifications of the `<EncryptedData>` part based on CBC and padding-properties.

These problems are described in detail in RFC 3218 [RFC3218].

The most effective countermeasure against the timing attack (2) is to generate a random secret key every time when the decrypted data was not PKCS#1-conformant. This way, the attacker would not get any timing side-channel.

Please note however that this is not a valid countermeasure against the specific modification of the `<EncryptedData>` described in part (3). The attacker could still use a few millions of requests to decrypt the encrypted symmetric key. Therefore, we recommend the usage of RSA-OAEP. RSA-OAEP also has a risk of a chosen ciphertext attack [OAEP-ATTACK] which can be mitigated in security library implementations.

6.1.3 Backwards Compatibility Attacks

Use of state-of-the-art and secure encryption algorithms such as RSA-OAEP and AES-GCM can become insecure when the adversary can force the server to process eavesdropped ciphertext with legacy algorithms such as RSA-PKCS#1 v1.5 or AES-CBC [XMLENC-BACKWARDS-COMP]:

1. The attacker may be able to break the security of an AES-GCM ciphertext if he is able to force the server to process the ciphertext with AES-CBC and the same symmetric key.
2. The attacker may be able to decrypt an RSA-OAEP ciphertext if he is able to force the server to process the ciphertext with RSA-PKCS#1 v1.5 and the same asymmetric key.
3. The attacker may be able to forge valid server signatures if the server decrypts RSA-PKCS#1 v1.5 ciphertexts and the signatures are computed with the same asymmetric key pair.

Accordingly, in situations where an attacker may be able to mount chosen-ciphertext attacks, we recommend the following to implementers:

1. Implementations SHOULD always use a different public key pair for data confidentiality and for data integrity functionality.
2. Implementations using symmetric keys SHOULD NOT use the same key material for different algorithms, even if serving the same purpose. Key derivation based on a single key and the algorithm identifier can be used to accomplish this, for example.
3. Implementations that plan to use the same symmetric key for both confidentiality and integrity functions SHOULD use it as the basis for a key derivation producing different keys for those functions.
4. Implementations SHOULD restrict algorithm usage to algorithms known to be secure in the face of chosen-ciphertext attacks (RSA-OAEP, AES-GCM). In that case, documents containing RSA-PKCS#1 v1.5 [XMLENC-PKCS15-ATTACK] and AES-CBC [XMLENC-CBC-ATTACK] ciphertexts SHOULD be rejected without decryption.

6.2 Relationship to XML Digital Signatures

The application of both encryption and digital signatures over portions of an XML document can make subsequent decryption and signature verification difficult. In particular, when verifying a signature one must know whether the signature was computed over the encrypted or unencrypted form of elements.

A separate, but important, issue is introducing cryptographic vulnerabilities when combining digital signatures and encryption over a common XML element. Hal Finney has suggested that encrypting digitally signed data, while leaving the digital signature in the clear, may allow plaintext guessing attacks. This vulnerability can be mitigated by using secure hashes and the nonces in the text being processed.

In accordance with the requirements document [XML-ENCRYPTION-REQ] the interaction of encryption and signing is an application issue and out of scope of the specification. However, we make the following recommendations:

1. When data is encrypted, any digest or signature over that data should be encrypted. This satisfies the first issue in that only those signatures that can be seen can be validated. It also addresses the possibility of a plaintext guessing vulnerability, though it may not be possible to identify (or even know of) all the signatures over a given piece of data.

2. Employ the "decrypt-except" signature transform [XMLENC-DECRYPT]. It works as follows: during signature transform processing, if you encounter a decrypt transform, decrypt all encrypted content in the document except for those excepted by an enumerated set of references.

Additionally, while the following warnings pertain to incorrect inferences by the user about the authenticity of information encrypted, applications should discourage user misapprehension by communicating clearly which information has integrity, or is authenticated, confidential, or non-repudiable when multiple processes (e.g., signature and encryption) and algorithms (e.g., symmetric and asymmetric) are used:

1. When an encrypted envelope contains a signature, the signature does not necessarily protect the authenticity or integrity of the ciphertext [Davis].

2. While the signature secures plaintext it only covers that which is signed, recipients of encrypted messages must not infer integrity or authenticity of other unsigned information (e.g., headers) within the encrypted envelope, see [XMLDSIG-CORE1], section 8.1.1 Only What is Signed is Secure].

6.3 Information Revealed

Where a symmetric key is shared amongst multiple recipients, that symmetric key should *only* be used for the data intended for *all* recipients; even if one recipient is not directed to information intended (exclusively) for another in the same symmetric key, the information might be discovered and decrypted.

Additionally, application designers should be careful not to reveal any information in parameters or algorithm identifiers (e.g., information in a URI) that weakens the encryption.

6.4 Nonce and IV (Initialization Value or Vector)

An undesirable characteristic of many encryption algorithms and/or their modes is that the same plaintext when encrypted with the same key has the same resulting ciphertext. While this is unsurprising, it invites various attacks which are mitigated by including an arbitrary and non-repeating (under a given key) data with the plaintext prior to encryption. In encryption chaining modes this data is the first to be encrypted and is consequently called the IV (initialization value or vector).

Different algorithms and modes have further requirements on the characteristic of this information (e.g., randomness and secrecy) that affect the features (e.g., confidentiality and integrity) and their resistance to attack.

Given that XML data is redundant (e.g., Unicode encodings and repeated tags) and that attackers may know the data's structure (e.g., DTDs and schemas) encryption algorithms must be carefully implemented and used in this regard.

For the Cipher Block Chaining (CBC) mode used by this specification, the IV must not be reused for any key and should be random, but it need not be secret. Additionally, under this mode an adversary modifying the IV can make a known change in the plain text after decryption. This attack can be avoided by securing the integrity of the plain text data, for example by signing it.

Note: CBC block encryption algorithms should not be used without consideration of possibly severe security risks.

For the Galois/Counter Mode (GCM) used by this specification, the IV must not be reused for any key and should be random, but it need not be secret.

6.5 Denial of Service

This specification permits recursive processing. For example, the following scenario is possible: `EncryptedKey` **A** requires `EncryptedKey` **B** to be decrypted, which itself requires `EncryptedKey` **A**! Or, an attacker might submit an `EncryptedData` for decryption that references network resources that are very large or continually redirected. Consequently, implementations should be able to restrict arbitrary recursion and the total amount of processing and networking resources a request can consume.

6.6 Unsafe Content

XML Encryption can be used to obscure, via encryption, content that applications (e.g., firewalls, virus detectors, etc.) consider unsafe (e.g., executable code, viruses, etc.). Consequently, such applications must consider encrypted content to be as unsafe as the unsafest content transported in its application context. Consequently, such applications may choose to (1) disallow such content, (2) require access to the decrypted form for inspection, or (3) ensure that arbitrary content can be safely processed by receiving applications.

6.7 Error Messages

Implementations SHOULD NOT provide detailed error responses related to security algorithm processing. Error messages should be limited to a generic error message to avoid providing information to a potential attacker related to the specifics of the algorithm implementation. For example, if an error occurs in decryption processing the error response should be a generic message providing no specifics on the details of the processing error.

6.8 Timing Attacks

It has been known for some time that it is feasible for an attacker to recover keys or cleartext by repeatedly sending chosen ciphertext and measuring the time required to process different requests with different types of errors. It has been demonstrated that attacks of this type are practical even when communicating over large and busy networks, especially if the receiver is willing to process large numbers of ciphertext blocks.

Implementers SHOULD ensure that distinct errors detected during security algorithm processing do not consume systematically different amounts of processing time from each other. Implementers SHOULD consult the technical literature for more details on specific attacks and recommended countermeasures.

Deployments SHOULD treat as suspect inputs when a large number of security algorithm processing errors are detected within a short period of time, especially in messages from the same origin.

6.9 CBC Block Encryption Vulnerability

Note: CBC block encryption algorithms should not be used without consideration of possibly severe security risks.

7. Conformance

An implementation is conformant to this specification if it successfully generates syntax according to the schema definitions and satisfies all MUST/REQUIRED/SHALL requirements, including algorithm support and processing. Processing requirements are specified over the roles of **decryptor**, **encryptor**, and their calling **application**.

8. XML Encryption Media Type

8.1 Introduction

XML Encryption Syntax and Processing (XMLENC-CORE1, this document) specifies a process for encrypting data and representing the result in XML. The data may be arbitrary data (including an XML document), an XML element, or XML element content. The result of encrypting data is an XML Encryption element which contains or references the cipher data.

The `application/xenc+xml` media type allows XML Encryption applications to identify encrypted documents. Additionally it allows applications cognizant of this media-type (even if they are not XML Encryption implementations) to note that the media type of the decrypted (original) object might be a type other than XML.

8.2 application/xenc+xml Registration

This is a media type registration as defined in Multipurpose Internet Mail Extensions (MIME) Part Four: Registration Procedures [MIME-REG]

Type name: application

Subtype name: xenc+xml

Required parameters: none

Optional parameters: charset

> The allowable and recommended values for, and interpretation of the charset parameter are identical to those given for 'application/xml' in section 3.2 of RFC 3023 [XML-MT].

Encoding considerations:

> The encoding considerations are identical to those given for 'application/xml' in section 3.2 of RFC 3023 [XML-MT].

Security considerations:

> See the (XMLENC-CORE1, this document) Security Considerations section.

Interoperability considerations: none

Published specification: (XMLENC-CORE1, this document)

Applications which use this media type:

> XML Encryption is device-, platform-, and vendor-neutral and is supported by a range of Web applications.

Additional Information:

> Magic number(s): none
>
> > Although no byte sequences can be counted on to consistently identify XML Encryption documents, there will be XML documents in which the root element's `QName`'s `LocalPart` is `'EncryptedData'` or '`EncryptedKey`' with an associated namespace name of 'http://www.w3.org/2001/04/xmlenc#'. The `application/xenc+xml` type name MUST only be used for data objects in which the root element is from the XML Encryption namespace. XML documents which contain these element types in places other than the root element can be described using facilities such as [XMLSCHEMA-1], [XMLSCHEMA-2].
>
> File extension(s): .xml
>
> Macintosh File Type Code(s): "TEXT"

Person & email address to contact for further information:

> World Wide Web Consortium <web-human at w3.org>

Intended usage: COMMON

Author/Change controller:

The XML Encryption specification is a work product of the World Wide Web Consortium (W3C) which has change control over the specification.

9. Schema

9.1 XSD Schema

XML Encryption Core Schema Instance
: xenc-schema.xsd

XML Encryption 1.1 Schema Instance
: xenc-schema11.xsd
: This schema document defines the additional material defined in XML Encryption 1.1.

Example (non-normative)
: enc-example.xml (not cryptographically valid but exercises much of the schema)

9.2 RNG Schema

This section is non-normative.

Non-normative RELAX NG schema [RELAXNG-SCHEMA] information is available in a separate document [XMLSEC-RELAXNG].

A. Reserved Algorithm Identifiers

This informative section outlines the definition and reserves identifiers for algorithms that have no requirements for implementation and have not been tested for interoperability.

A.1 AES KeyWrap with Padding

This section is non-normative.

Identifiers:

http://www.w3.org/2009/xmlenc11#kw-aes-128-pad
http://www.w3.org/2009/xmlenc11#kw-aes-192-pad
http://www.w3.org/2009/xmlenc11#kw-aes-256-pad

These identifiers are reserved for symmetric key wrapping using the AES key wrap with padding algorithm with a 128, 192, and 256 bit AES key encrypting key, respectively. Implementation of AES key wrap with padding is defined in [AES-WRAP-PAD]. The algorithm is defined for inputs between 9 and 2^32 octets. Unlike the unpadded AES Key Wrap algorithm, the input length is not constrained to multiples of 64 bits (8 octets).

Note that the wrapped key will be distinct from the one generated by the unpadded AES Key Wrap algorithm, even if the input length is a multiple of 64 bits.

B. References

Dated references below are to the latest known or appropriate edition of the referenced work. The referenced works may be subject to revision, and conformant implementations may follow, and are encouraged to investigate the appropriateness of following, some or all more recent editions or replacements of the works cited. It is in each case implementation-defined which editions are supported.

B.1 Normative references

[AES]
NIST FIPS 197: Advanced Encryption Standard (AES). November 2001. URL: http://csrc.nist.gov/publications/fips/fips197/fips-197.pdf

[AES-WRAP]
J. Schaad; R. Housley. *RFC3394: Advanced Encryption Standard (AES) Key Wrap Algorithm*. September 2002. IETF Informational RFC. URL: http://www.ietf.org/rfc/rfc3394.txt

[AES-WRAP-PAD]
R. Housley; M. Dworkin. *RFC 5649: Advanced Encryption Standard (AES) Key Wrap with Padding Algorithm*. August 2009. IETF Informational RFC. URL: http://www.ietf.org/rfc/rfc5649.txt

[ANSI-X9-44-2007]
ANSI X9.44-2007: Key Establishment Using Integer Factorization Cryptography. URL: http://webstore.ansi.org/RecordDetail.aspx?sku=ANSI+X9.44-2007

[CMS-WRAP]
R. Housley. *RFC3217: Triple-DES and R2 Key Wrapping*. December 2001. IETF Informational RFC. URL: http://www.ietf.org/rfc/rfc3217.txt

[DES]
NIST FIPS 46-3: Data Encryption Standard (DES) . October 1999. URL: http://csrc.nist.gov/publications/fips/fips46-3/fips46-3.pdf

[ESDH]
E. Rescorla. *Diffie-Hellman Key Agreement Method.*. IETF RFC 2631 Standards Track, 1999. URL: http://www.ietf.org/rfc/rfc2631.txt

[EXI]
Takuki Kamiya; John Schneider. *Efficient XML Interchange (EXI) Format 1.0*. 8 December 2009. W3C Candidate Recommendation. URL: http://www.w3.org/TR/2009/CR-exi-20091208/

[FIPS-180-3]
FIPS PUB 180-3 Secure Hash Standard. U.S. Department of Commerce/National Institute of Standards and Technology. URL: http://csrc.nist.gov/publications/fips/fips180-3/fips180-3_final.pdf

[FIPS-186-3]
FIPS PUB 186-3: Digital Signature Standard (DSS). June 2009. U.S. Department of Commerce/National Institute of Standards and Technology. URL: http://csrc.nist.gov/publications/fips/fips186-3/fips_186-3.pdf

[HMAC]
H. Krawczyk, M. Bellare, R. Canetti. *HMAC: Keyed-Hashing for Message Authentication*. February 1997. IETF RFC 2104. URL: http://www.ietf.org/rfc/rfc2104.txt

[NFC]
M. Davis, Ken Whistler. *TR15, Unicode Normalization Forms.*. 17 September 2010, URL: http://www.unicode.org/reports/tr15/

[PKCS1]
J. Jonsson and B. Kaliski. *Public-Key Cryptography Standards (PKCS) #1: RSA Cryptography Specifications Version 2.1.* RFC 3447 (Informational), February 2003. URL: http://www.ietf.org/rfc/rfc3447.txt

[PKCS5]
B. Kaliski. *PKCS #5 v2.0: Password-Based Cryptography Standard*. September 2000. IETF RFC 2898. URL: http://www.ietf.org/rfc/rfc2898.txt

[PKCS5Amd1]
PKCS #5 v2.0 Amendment 1: XML Schema for Password-Based Cryptography RSA Laboratories, March 2007. URL: ftp://ftp.rsasecurity.com/pub/pkcs/pkcs-5v2/pkcs-5v2-0a1.pdf

[RANDOM]
D. Eastlake, S. Crocker, J. Schiller. *Randomness Recommendations for Security.*. IETF RFC 4086. June 2005. URL: http://www.ietf.org/rfc/rfc4086.txt

[RFC2045]
N. Freed and N. Borenstein. *Multipurpose Internet Mail Extensions (MIME) Part One: Format of Internet Message Bodies.* November 1996. URL: http://www.ietf.org/rfc/rfc2045.txt

[RFC2119]
S. Bradner. *Key words for use in RFCs to Indicate Requirement Levels.* March 1997. Internet RFC 2119. URL: http://www.ietf.org/rfc/rfc2119.txt

[RFC4055]
J. Schaad, B. Kaliski, R. Housley. *Additional Algorithms and Identifiers for RSA Cryptography for use in the Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile*. June 2005. IETF RFC 4055. URL: http://www.ietf.org/rfc/rfc4055.txt

[RIPEMD-160]
B. Preneel, A. Bosselaers, and H. Dobbertin. *The Cryptographic Hash Function RIPEMD-160*. CryptoBytes, Volume 3, Number 2. pp. 9-14, RSA Laboratories 1997. URL: http://www.cosic.esat.kuleuven.be/publications/article-317.pdf
[SP800-38D]
M. Dworkin. *NIST Special Publication 800-38D: Recommendation for Block Cipher Modes of Operation: Galois/Counter Mode (GCM) and GMAC*. November 2007 URL: http://csrc.nist.gov/publications/nistpubs/800-38D/SP-800-38D.pdf
[SP800-56A]
NIST Special Publication 800-56A: Recommendation for Pair-Wise Key Establishment Schemes Using Discrete Logarithm Cryptography (Revised). March 2007 URL: http://csrc.nist.gov/publications/nistpubs/800-56A/SP800-56A_Revision1_Mar08-2007.pdf
[SP800-67]
Recommendation for the Triple Data Encryption Algorithm (TDEA) Block Cipher, Revised January 2012. SP-800-67 Revision 1. U.S. Department of Commerce/National Institute of Standards and Technology. URL: http://csrc.nist.gov/publications/nistpubs/800-67-Rev1/SP-800-67-Rev1.pdf
[URI]
T. Berners-Lee; R. Fielding; L. Masinter. *Uniform Resource Identifiers (URI): generic syntax*. January 2005. RFC 3986. URL: http://www.ietf.org/rfc/rfc3986.txt
[XML-ENCRYPTION-REQ]
Joseph Reagle. *XML Encryption Requirements*. 4 March 2002. W3C Note. URL: http://www.w3.org/TR/2002/NOTE-xml-encryption-req-20020304
[XML-NAMES]
Richard Tobin et al. *Namespaces in XML 1.0 (Third Edition)*. 8 December 2009. W3C Recommendation. URL: http://www.w3.org/TR/2009/REC-xml-names-20091208/
[XML10]
C. M. Sperberg-McQueen et al. *Extensible Markup Language (XML) 1.0 (Fifth Edition)*. 26 November 2008. W3C Recommendation. URL: http://www.w3.org/TR/2008/REC-xml-20081126/
[XMLDSIG-CORE1]
D. Eastlake; J. Reagle; D. Solo; F. Hirsch; T. Roessler; K. Yiu. *XML Signature Syntax and Processing Version 1.1*. 11 April 2013. W3C Recommendation. URL: http://www.w3.org/TR/2013/REC-xmldsig-core1-20130411/
[XMLSCHEMA-1]
Henry S. Thompson et al. *XML Schema Part 1: Structures Second Edition*. 28 October 2004. W3C Recommendation. URL: http://www.w3.org/TR/2004/REC-xmlschema-1-20041028/
[XMLSCHEMA-2]
Paul V. Biron; Ashok Malhotra. *XML Schema Part 2: Datatypes Second Edition*. 28 October 2004. W3C Recommendation. URL: http://www.w3.org/TR/2004/REC-xmlschema-2-20041028/
[XPATH]
James Clark; Steven DeRose. *XML Path Language (XPath) Version 1.0*. 16 November 1999. W3C Recommendation. URL: http://www.w3.org/TR/1999/REC-xpath-19991116/

B.2 Informative references

[Davis]
Defective Sign & Encrypt in S/MIME, PKCS#7, MOSS, PEM, PGP, and XML. D. Davis. USENIX Annual Technical Conference. 2001. URL: http://www.usenix.org/publications/library/proceedings/usenix01/davis.html
[ECC-ALGS]
D. McGrew; K. Igoe; M. Salter. *RFC 6090: Fundamental Elliptic Curve Cryptography Algorithms*. February 2011. IETF Informational RFC. URL: http://www.rfc-editor.org/rfc/rfc6090.txt
[MIME-REG]
N. Freed, J. Klensin. *RFC 4289: Multipurpose Internet Mail Extensions (MIME) Part Four: Registration Procedures*. December 2005. Best Current Practice. URL: http://www.ietf.org/rfc/rfc4289.txt
[OAEP-ATTACK]
Manger, James. *A Chosen Ciphertext Attack on RSA Optimal Asymmetric Encryption Padding (OAEP) as Standardized in PKCS #1 v2.0*. URL: http://archiv.infsec.ethz.ch/education/fs08/secsem/Manger01.pdf
[RELAXNG-SCHEMA]
Information technology -- Document Schema Definition Language (DSDL) -- Part 2: Regular-grammar-based validation -- RELAX NG. ISO/IEC 19757-2:2008. URL: http://standards.iso.org/ittf/PubliclyAvailableStandards/c052348_ISO_IEC_19757-2_2008(E).zip
[RFC3218]
E. Rescorla. *Preventing the Million Message Attack on Cryptographic Message Syntax (RFC 3218)*. January 2002. RFC. URL: http://www.rfc-editor.org/rfc/rfc3218.txt
[SHA-1-Analysis]
McDonald, C., Hawkes, P., and J. Pieprzyk. *SHA-1 collisions now 2^{52}*. EuroCrypt 2009 Rump session. URL: http://eurocrypt2009rump.cr.yp.to/837a0a8086fa6ca714249409ddfae43d.pdf
[SHA-1-Collisions]
X. Wang, Y.L. Yin, H. Yu. *Finding Collisions in the Full SHA-1*. In Shoup, V., editor, Advances in Cryptology - CRYPTO 2005, 25th Annual International Cryptology Conference, Santa Barbara, California, USA, August 14-18, 2005, Proceedings, volume 3621 of LNCS, pages 17–36. Springer, 2005. URL: http://people.csail.mit.edu/yiqun/SHA1AttackProceedingVersion.pdf (also published in http://www.springerlink.com/content/26vljj3xhc28ux5m/)
[Tobin]
R. Tobin. *Infoset for external entities.* 2000. URL: http://lists.w3.org/Archives/Member/w3c-xml-core-wg/2000OctDec/0054 [XML Core mailing list, W3C Member Only].
[XML-C14N]
John Boyer. *Canonical XML Version 1.0*. 15 March 2001. W3C Recommendation. URL: http://www.w3.org/TR/2001/REC-xml-c14n-20010315
[XML-C14N11]
John Boyer; Glenn Marcy. *Canonical XML Version 1.1*. 2 May 2008. W3C Recommendation. URL: http://www.w3.org/TR/2008/REC-xml-c14n11-20080502/
[XML-EXC-C14N]
Donald E. Eastlake 3rd; Joseph Reagle; John Boyer. *Exclusive XML Canonicalization Version 1.0*. 18 July 2002. W3C Recommendation. URL: http://www.w3.org/TR/2002/REC-xml-exc-c14n-20020718/
[XML-INFOSET]
John Cowan; Richard Tobin. *XML Information Set (Second Edition)*. 4 February 2004. W3C Recommendation. URL: http://www.w3.org/TR/2004/REC-xml-infoset-20040204/
[XML-MT]

M. Murata, S. St.Laurent, D. Kohn. *XML Media Types*. IETF RFC 3023. URL: http://www.ietf.org/rfc/rfc3023.txt.

[XMLBASE]
Jonathan Marsh; Richard Tobin. *XML Base (Second Edition)*. 28 January 2009. W3C Recommendation. URL: http://www.w3.org/TR/2009/REC-xmlbase-20090128/

[XMLENC-BACKWARDS-COMP]
Tibor Jager; Kenneth G. Paterson; Juraj Somorovsky. *One Bad Apple: Backwards Compatibility Attacks on State-of-the-Art Cryptography*. 2013. URL: http://www.nds.ruhr-uni-bochum.de/research/publications/backwards-compatibility/

[XMLENC-CBC-ATTACK]
Tibor Jager; Juraj Somorovsky. *How to Break XML Encryption*. 17-21 October 2011. CCS'11, ACM. URL: http://www.nds.ruhr-uni-bochum.de/research/publications/breaking-xml-encryption/

[XMLENC-CBC-ATTACK-COUNTERMEASURES]
Juraj Somorovsky; Jörg Schwenk. *Análisis técnico de contramedidas contra ataques al cifrado XML, o simplemente otra motivación para el cifrado autenticado* . 2011. URL: http://www.w3.org/2008/xmlsec/papers/xmlEncCountermeasuresW3C.pdf

[XMLENC-CORE1-CHGS]
Federico Hirsch. *Explicación funcional de XML Encryption 1.1* . 11 de abril de 2013. Nota del grupo de trabajo del W3C. URL: http://www.w3.org/TR/2013/NOTE-xmlenc-core1-explain-20130411/

[XMLENC-DECRIPT]
Takeshi Imamura; Merlín Hughes; Hiroshi Maruyama. *Transformación de descifrado para firma XML* . 10 de diciembre de 2002. Recomendación del W3C. URL: http://www.w3.org/TR/2002/REC-xmlenc-decrypt-20021210

[XMLENC-PKCS15-ATAQUE]
Tibor Jäger; Sebastián Schinzel; Juraj Somorovsky. *El ataque de Bleichenbacher ataca de nuevo: romper PKCS#1.5 en cifrado XML* . 2012. URL: http://www.nds.rub.de/research/publications/breaking-xml-encryption-pkcs15.pdf

[XMLSEC-RELAJARSE]
Makoto Murata; Federico Hirsch. *Seguridad XML Esquemas RELAX NG* . 11 de abril de 2013. Nota del grupo de trabajo del W3C. URL: http://www.w3.org/TR/2013/NOTE-xmlsec-rngschema-20130411/

[XMLSEC11-REQS]
Federico Hirsch; Tomás Roessler. *Requisitos de seguridad XML 1.1 y consideraciones de diseño* . 11 de abril de 2013. Nota del grupo de trabajo del W3C. URL: http://www.w3.org/TR/2013/NOTE-xmlsec-reqs-20130411/